

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problem Mailbox.**

- 301 -

```

{
    PullRightMenuWidget    prw=(PullRightMenuWidget)w;
    SmeObject    entry=prw->simple_menu.entry_set;
    SmeObjectClass    class;

    if ((entry == NULL) || !XtIsSensitive((Widget)entry))return;
    if (event->type != LeaveNotify && event->type != EnterNotify) {
        XtAppError(XtWidgetToApplicationContext(w),
            "pull() action should only be used with XCrossing events.");
        return;
    }
    if (None != event->xcrossing.subwindow) return;
    if (event->xcrossing.y < 0 || event->xcrossing.y > prw->core.height) {
        Unhighlight(w,event,params,num_params);
        return;
    };
    if (event->xcrossing.x < 0) {
        if (XtIsSubclass(XtParent(w),pullRightMenuWidgetClass)) XtPopdown(w);
        return;
    };
    class=(SmeObjectClass)entry->object.widget_class;
    if (event->xcrossing.x > prw->core.width &&
        XtIsSubclass(entry,smeBSBprObjectClass)) (class->sme_class.notify)((Widget)entry);
    else Unhighlight(w,event,params,num_params);
}

/*  Function Name: Execute
*  Description: Determines notify action on basis of SmeObject.
*  Arguments: w - the pull right menu widget.
*              event - the notify-type event that caused this action.
*              params, num_params - ** NOT USED **
*  Returns: none

```

- 302 -

*/

```
static void Execute(w, event, params, num_params)
```

```
Widget      w;
```

```
XEvent      *event;
```

```
String *params;
```

```
Cardinal     *num_params;
```

```
{
```

```
    PullRightMenuWidget    prw = (PullRightMenuWidget)w;
```

```
    SmeObject    entry = prw->simple_menu.entry_set;
```

```
    SmeObjectClass    class;
```

```
    Widget    shell;
```

```
    Dprintf("Execute\n");
```

```
    for(shell = w:XtIsSubclass(shell, pullRightMenuWidgetClass); shell = XtParent(shell))
```

```
{
```

```
    XawSimpleMenuClearActiveEntry(shell);
```

```
    XtPopdown(shell);
```

```
};
```

```
if
```

```
((entry == GetEventEntry(w, event)) && (entry != NULL) && XtIsSensitive((Widget)entry)) {
```

```
    class = (SmeObjectClass)entry->object.widget_class;
```

```
    if (XtIsSubclass(entry, smeBSBObjectClass))
```

```
(class->sme_class.notify)((Widget)entry);
```

```
};
```

```
}
```

```
.....
```

- 303 -

*

* Public Functions.

*

*****/

```

/*  Function Name: XawPullRightMenuAddGlobalActions
*   Description: adds the global actions to the simple menu widget.
*   Arguments: app_con - the appcontext.
*   Returns: none.
*/

```

void

XawPullRightMenuAddGlobalActions(app_con)

XtAppContext app_con;

{

XtInitializeWidgetClass(pullRightMenuWidgetClass);

XmuCallInitializers(app_con);

}

/*****

*

* Private Functions.

*

*****/

```

/*  Function Name: CreateLabel
*   Description: Creates a the menu label.
*   Arguments: w - the smw widget.
*   Returns: none.

```

*

```

* Creates the label object and makes sure it is the first child in
* in the list.

```


- 304 -

*/

static void

CreateLabel(w)

Widget w;

{

SimpleMenuWidget smw = (SimpleMenuWidget) w;

register Widget * child, * next_child;

register int i;

Arg args[2];

if ((smw->simple_menu.label_string == NULL) ||

(smw->simple_menu.label != NULL)) {

char error_buf[BUFSIZ];

sprintf(error_buf, "Xaw Simple Menu Widget: %s or %s, %s",

"label string is NULL", "label already exists",

"no label is being created.");

XtAppWarning(XtWidgetToApplicationContext(w), error_buf);

return;

}

XtSetArg(args[0], XtNlabel, smw->simple_menu.label_string);

XtSetArg(args[1], XtNjustify, XtJustifyCenter);

smw->simple_menu.label = (SmeObject)

XtCreateManagedWidget("menuLabel",

smw->simple_menu.label_class, w,

args, TWO);

next_child = NULL;

for (child = smw->composite.children + smw->composite.num_children,

i = smw->composite.num_children ; i > 0 ; i--, child--) {

- 305 -

```

    if (next_child != NULL)
        *next_child = *child;
    next_child = child;
}
*child = (Widget) smw->simple_menu.label;
}

```

```

/*    Function Name: Layout
*    Description: lays the menu entries out all nice and neat.
*    Arguments: w - See below (+++)
*               width_ret, height_ret - The returned width and
*               height values.
*    Returns: none.
*
* if width == NULL || height == NULL then it assumes the you do not care
* about the return values, and just want a relayout.
*
* if this is not the case then it will set width_ret and height_ret
* to be width and height that the child would get if it were layed out
* at this time.
*
* +++ "w" can be the simple menu widget or any of its object children.
*/

```

```
static void
```

```
Layout(w, width_ret, height_ret)
```

```
Widget w;
```

```
Dimension *width_ret, *height_ret;
```

```
{
```

```
    SmeObject current_entry, *entry;
```

```
    SimpleMenuWidget smw;
```

```
    Dimension width, height;
```

- 306 -

```
Boolean do_layout = ((height_ret == NULL) || (width_ret == NULL));
```

```
Boolean allow_change_size;
```

```
height = 0;
```

```
if ( XtIsSubclass(w, puliRightMenuWidgetClass) ) {
```

```
    smw = (SimpleMenuWidget) w;
```

```
    current_entry = NULL;
```

```
}
```

```
else {
```

```
    smw = (SimpleMenuWidget) XtParent(w);
```

```
    current_entry = (SmeObject) w;
```

```
}
```

```
allow_change_size = (!XtIsRealized((Widget)smw) ||  
    (smw->shell.allow_shell_resize));
```

```
if ( smw->simple_menu.menu_height )
```

```
    height = smw->core.height;
```

```
else
```

```
    if (do_layout) {
```

```
        height = smw->simple_menu.top_margin;
```

```
        ForAllChildren(smw, entry) {
```

```
            if (!XtIsManaged( (Widget) *entry)) continue;
```

```
            if ( (smw->simple_menu.row_height != 0) &&
```

```
                (*entry != smw->simple_menu.label) )
```

```
                (*entry)->rectangle.height = smw->simple_menu.row_height;
```

```
            (*entry)->rectangle.y = height;
```

```
            (*entry)->rectangle.x = 0;
```

```
            height += (*entry)->rectangle.height;
```

```
        }
```

- 307 -

```
        height += smw->simple_menu.bottom_margin;
    }
    else {
        if ((smw->simple_menu.row_height != 0) &&
            (current_entry != smw->simple_menu.label) )
            height = smw->simple_menu.row_height;
    }

    if (smw->simple_menu.menu_width)
        width = smw->core.width;
    else if ( allow_change_size )
        width = GetMenuWidth((Widget) smw, (Widget) current_entry);
    else
        width = smw->core.width;

    if (do_layout) {
        ForAllChildren(smw, entry)
            if (XtIsManaged( (Widget) *entry))
                (*entry)->rectangle.width = width;

        if (allow_change_size)
            MakeSetValuesRequest((Widget) smw, width, height);
    }
    else {
        *width_ret = width;
        if (height != 0)
            *height_ret = height;
    }
}
```

/* Function Name: AddPositionAction

* Description: Adds the XawPositionSimpleMenu action to the global

- 308 -

```

*           action list for this appcon.
*   Arguments: app_con - the application context for this app.
*           data - NOT USED.
*   Returns: none.
*/

```

```

/* ARGSUSED */

```

```

static void

```

```

AddPositionAction(app_con, data)

```

```

XtAppContext app_con;

```

```

caddr_t data;

```

```

{
    static XtActionsRec pos_action[] = {
        { "XawPositionSimpleMenu", PositionMenuAction },
    };

    XtAppAddActions(app_con, pos_action, XtNumber(pos_action));
}

```

```

/*   Function Name: FindMenu

```

```

*   Description: Find the menu give a name and reference widget.
*   Arguments: widget - reference widget.
*           name  - the menu widget's name.
*   Returns: the menu widget or NULL.
*/

```

```

static Widget

```

```

FindMenu(widget, name)

```

```

Widget widget;

```

```

String name;

```

```

{
    register Widget w, menu;

```

- 309 -

```

for ( w = widget ; w != NULL ; w = XtParent(w) )
    if ( (menu = XtNameToWidget(w, name)) != NULL )
        return(menu);
return(NULL);
}

```

```

/*    Function Name: PositionMenu
 *    Description: Places the menu
 *    Arguments: w - the simple menu widget.
 *               location - a pointer the the position or NULL.
 *    Returns: none.
 */

```

```

static void
PositionMenu(w, location)
Widget w;
XPoint * location;
{
    SimpleMenuWidget smw = (SimpleMenuWidget) w;
    SmeObject entry;
    XPoint t_point;
    static void MoveMenu();

    if (location == NULL) {
        Window junk1, junk2;
        int root_x, root_y, junkX, junkY;
        unsigned int junkM;

        location = &t_point;
        if (XQueryPointer(XtDisplay(w), XtWindow(w), &junk1, &junk2,
            &root_x, &root_y, &junkX, &junkY, &junkM) == FALSE) {

```

- 310 -

```

char error_buf[BUFSIZ];
sprintf(error_buf, "%s %s", "Xaw - SimpleMenuWidget:",
        "Could not find location of mouse pointer");
XtAppWarning(XtWidgetToApplicationContext(w), error_buf);
return;
}
location->x = (short) root_x;
location->y = (short) root_y;
}

/*
 * The width will not be correct unless it is realized.
 */

XtRealizeWidget(w);

location->x -= (Position) w->core.width/2;

if (smw->simple_menu.popup_entry == NULL)
    entry = smw->simple_menu.label;
else
    entry = smw->simple_menu.popup_entry;

if (entry != NULL)
    location->y -= entry->rectangle.y + entry->rectangle.height/2;

MoveMenu(w, (Position) location->x, (Position) location->y);
}

/*  Function Name: MoveMenu
 *  Description: Actually moves the menu, may force it to
 *               to be fully visible if menu_on_screen is TRUE.

```

- 311 -

- * Arguments: w - the simple menu widget.
- * x, y - the current location of the widget.
- * Returns: none
- */

static void

MoveMenu(w, x, y)

Widget w;

Position x, y;

{

Arg arglist[2];

Cardinal num_args = 0;

SimpleMenuWidget smw = (SimpleMenuWidget) w;

if (smw->simple_menu.menu_on_screen) {

int width = w->core.width + 2 * w->core.border_width;

int height = w->core.height + 2 * w->core.border_width;

if (x < 0)

 x = 0;

else {

 int scr_width = WidthOfScreen(XtScreen(w));

 if (x + width > scr_width)

 x = scr_width - width;

}

if (y < 0)

 y = 0;

else {

 int scr_height = HeightOfScreen(XtScreen(w));

 if (y + height > scr_height)

 y = scr_height - height;

- 312 -

```

    }
}

XtSetArg(arglist[num_args], XtNx, x); num_args++;
XtSetArg(arglist[num_args], XtNy, y); num_args++;
XtSetValues(w, arglist, num_args);
}

```

```

/*  Function Name: ChangeCursorOnGrab
*   Description: Changes the cursor on the active grab to the one
*               specified in our resource list.
*   Arguments: w - the widget.
*               junk, garbage - ** NOT USED **.
*   Returns: None.
*/

```

```

/* ARGSUSED */

```

```

static void

```

```

ChangeCursorOnGrab(w, junk, garbage)

```

```

Widget w;

```

```

caddr_t junk, garbage;

```

```

{

```

```

    SimpleMenuWidget smw = (SimpleMenuWidget) w;

```

```

/*

```

```

* The event mask here is what is currently in the MIT implementation.
* There really needs to be a way to get the value of the mask out
* of the toolkit (CDP 5/26/89).

```

```

*/

```

```

XChangeActivePointerGrab(XtDisplay(w), ButtonPressMask | ButtonReleaseMask,
    smw->simple_menu.cursor, CurrentTime);

```

- 313 -

}

```

/*  Function Name: MakeSetValuesRequest
*   Description: Makes a (possibly recursive) call to SetValues,
*               I take great pains to not go into an infinite loop.
*   Arguments: w - the simple menu widget.
*               width, height - the size of the ask for.
*   Returns: none
*/

```

```
static void
```

```
MakeSetValuesRequest(w, width, height)
```

```
Widget w;
```

```
Dimension width, height;
```

```
{
```

```
SimpleMenuWidget smw = (SimpleMenuWidget) w;
```

```
Arg arglist[2];
```

```
Cardinal num_args = (Cardinal) 0;
```

```
if ( !smw->simple_menu.recursive_set_values ) {
```

```
    if ( (smw->core.width != width) || (smw->core.height != height) ) {
```

```
        smw->simple_menu.recursive_set_values = TRUE;
```

```
        XtSetArg(arglist[num_args], XtNwidth, width);  num_args++;
```

```
        XtSetArg(arglist[num_args], XtNheight, height); num_args++;
```

```
        XtSetValues(w, arglist, num_args);
```

```
    }
```

```
    else if (XtIsRealized( (Widget) smw))
```

```
        Redisplay((Widget) smw, (XEvent *) NULL, (Region) NULL);
```

```
}
```

```
smw->simple_menu.recursive_set_values = FALSE;
```

```
}
```

- 314 -

```
/*  Function Name: GetMenuWidth
 *   Description: Sets the length of the widest entry in pixels.
 *   Arguments: w - the simple menu widget.
 *   Returns: width of menu.
 */
```

```
static Dimension
```

```
GetMenuWidth(w, w_ent)
```

```
Widget w, w_ent;
```

```
{
```

```
    SmeObject cur_entry = (SmeObject) w_ent;
```

```
    SimpleMenuWidget smw = (SimpleMenuWidget) w;
```

```
    Dimension width, widest = (Dimension) 0;
```

```
    SmeObject * entry;
```

```
    if ( smw->simple_menu.menu_width )
```

```
        return(smw->core.width);
```

```
    ForAllChildren(smw, entry) {
```

```
        XtWidgetGeometry preferred;
```

```
        if (!XtIsManaged( (Widget) *entry)) continue;
```

```
        if (*entry != cur_entry) {
```

```
            XtQueryGeometry(*entry, NULL, &preferred);
```

```
            if (preferred.request_mode & CWWidth)
```

```
                width = preferred.width;
```

```
            else
```

```
                width = (*entry)->rectangle.width;
```

```
        }
```

```
    else
```

- 315 -

```

        width = (*entry)->rectangle.width;

        if ( width > widest )
            widest = width;
    }

    return(widest);
}

/*  Function Name: GetMenuHeight
 *  Description: Sets the length of the widest entry in pixels.
 *  Arguments: w - the simple menu widget.
 *  Returns: width of menu.
 */

static Dimension
GetMenuHeight(w)
Widget w;
{
    SimpleMenuWidget smw = (SimpleMenuWidget) w;
    SmeObject * entry;
    Dimension height;

    if (smw->simple_menu.menu_height)
        return(smw->core.height);

    height = smw->simple_menu.top_margin + smw->simple_menu.bottom_margin;

    if (smw->simple_menu.row_height == 0)
        ForAllChildren(smw, entry)
            if (XtIsManaged ((Widget) *entry))
                height += (*entry)->rectangle.height;

```

- 316 -

```
else
    height += smw->simple_menu.row_height * smw->composite.num_children;

return(height);
}

/*  Function Name: GetEventEntry
 *   Description: Gets an entry given an event that has X and Y coords.
 *   Arguments: w - the simple menu widget.
 *              event - the event.
 *   Returns: the entry that this point is in.
 */

static SmeObject
GetEventEntry(w, event)
Widget w;
XEvent * event;
{
    Position x_loc, y_loc;
    SimpleMenuWidget smw = (SimpleMenuWidget) w;
    SmeObject * entry;

    switch (event->type) {
    case MotionNotify:
        x_loc = event->xmotion.x;
        y_loc = event->xmotion.y;
        break;
    case EnterNotify:
    case LeaveNotify:
        x_loc = event->xcrossing.x;
        y_loc = event->xcrossing.y;
        break;
```

- 317 -

```
case ButtonPress:
```

```
case ButtonRelease:
```

```
    x_loc = event->xbunon.x;
```

```
    y_loc = event->xbunon.y;
```

```
    break;
```

```
default:
```

```
    XtAppError(XtWidgetToApplicationContext(w),
```

```
                "Unknown event type in GetEventEntry().");
```

```
    break;
```

```
}
```

```
if ( (x_loc < 0) || (x_loc >= smw->core.width) || (y_loc < 0) ||
```

```
    (y_loc >= smw->core.height) )
```

```
    return(NULL);
```

```
ForAllChildren(smw, entry) {
```

```
    if (!XtIsManaged ((Widget) *entry)) continue;
```

```
    if ( ((*entry)->rectangle.y < y_loc) &&
```

```
        ((*entry)->rectangle.y + (*entry)->rectangle.height > y_loc) )
```

```
        if ( *entry == smw->simple_menu.label )
```

```
            return(NULL);    /* cannot select the label. */
```

```
        else
```

```
            return(*entry);
```

```
}
```

```
return(NULL);
```

```
}
```

- 318 -

source/Select.c

```
/*
 * Selection from list widget
 *
 */

#include    "../include/xwave.h"

void Select(w,closure,call_data)

Widget      w;
caddr_t      closure, call_data;

{
    Selection    sel = (Selection)closure;
    Widget      button = FindWidget(sel->button,w),
               shell = ShellWidget(sel->name,button,SW_below,NULL,NULL),
               form = FormatWidget("sel_form",shell), list_widget, widgets[3];
    String *list = (sel->list_proc)();
    FormItem     items[] = {
        {"sel_cancel","close",0,0,FW_icon,NULL},
        {"sel_label",(String)sel->action_name,1,0,FW_label,NULL},
        {"sel_view",NULL,0,2,FW_view,NULL},
    };
    XtCallbackRec list_calls[] = {
        {Destroy,(caddr_t)shell},
        {sel->action_proc,sel->action_closure},
        {NULL,NULL},
    }, callbacks[] = {
```

- 319 -

```
        {Destroy,(caddr_t)shell},  
        {NULL,NULL},  
    };  
    Arg    args[1];  
  
    FillForm(form,THREE,items,widgets,callbacks);  
    XtSetArg(args[0],XtNlist,list);  
  
    list_widget=XtCreateManagedWidget("sel_list",listWidgetClass,widgets[2],args,ONE);  
    XtAddCallbacks(list_widget,XtNcallback,list_calls);  
    XtPopup(shell,XtGrabExclusive);  
}
```


- 320 -

source/SmeBSBpr.c

#if (!defined(lint) && !defined(SABER))

static char Xrcsid[] = "\$XConsortium: SmeBSB.c,v 1.9 89/12/13 15:42:48 kit Exp \$";

#endif

/*

* Copyright 1989 Massachusetts Institute of Technology

*

- * Permission to use, copy, modify, distribute, and sell this software and its
- * documentation for any purpose is hereby granted without fee, provided that
- * the above copyright notice appear in all copies and that both that
- * copyright notice and this permission notice appear in supporting
- * documentation, and that the name of M.I.T. not be used in advertising or
- * publicity pertaining to distribution of the software without specific,
- * written prior permission. M.I.T. makes no representations about the
- * suitability of this software for any purpose. It is provided "as is"
- * without express or implied warranty.
- *
- * M.I.T. DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
- INCLUDING ALL
- * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO
- EVENT SHALL M.I.T.
- * BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES
- OR ANY DAMAGES
- * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
- WHETHER IN AN ACTION
- * OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT
- OF OR IN
- * CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

- 321 -

*/

/*

* SmeBSBpr.c - Source code file for BSB pull-right Menu Entry object.

*

*/

#include <stdio.h>

#include <X11/IntrinsicP.h>

#include <X11/StringDefs.h>

#include <X11/Xmu/Drawing.h>

#include <X11/Xaw/XawInit.h>

#include <X11/Xaw/SimpleMenu.h>

#include "SmeBSBprP.h"

#include <X11/Xaw/Cardinals.h>

#define ONE_HUNDRED 100

#define offset(field) XtOffset(SmeBSBprObject, sme_bsb.field)

static XtResource resources[] = {

{XtNlabel, XtCLabel, XtRString, sizeof(String),
offset(label), XtRString, NULL},{XtNvertSpace, XtCVertSpace, XtRInt, sizeof(int),
offset(vert_space), XtRImmediate, (caddr_t) 25},{XtNleftBitmap, XtCLeftBitmap, XtRPixmap, sizeof(Pixmap),
offset(left_bitmap), XtRImmediate, (caddr_t)None},{XtNjustify, XtCJustify, XtRJustify, sizeof(XtJustify),
offset(justify), XtRImmediate, (caddr_t) XtJustifyLeft},

{XtNrightBitmap, XtCRightBitmap, XtRPixmap, sizeof(Pixmap),

- 322 -

```

    offset(right_bitmap), XtRImmediate, (caddr_t)None},
    {XtNleftMargin, XtCHorizontalMargins, XtRDimension, sizeof(Dimension),
      offset(left_margin), XtRImmediate, (caddr_t) 4},
    {XtNrightMargin, XtCHorizontalMargins, XtRDimension, sizeof(Dimension),
      offset(right_margin), XtRImmediate, (caddr_t) 4},
    {XtNforeground, XtCForeground, XtRPixel, sizeof(Pixel),
      offset(foreground), XtRString, "XtDefaultForeground"},
    {XtNfont, XtCFont, XtRFontStruct, sizeof(XFontStruct *),
      offset(font), XtRString, "XtDefaultFont"},
    {XtNmenuName, XtCMenuName, XtRString, sizeof(String),
      offset(menu_name), XtRString, (caddr_t)"menu"},
};
#undef offset

/*
 * Semi Public function definitions.
 */

static void Redisplay(), Destroy(), Initialize(), FlipColors(), PopupMenu();
static void ClassInitialize();
static Boolean SetValues();
static XtGeometryResult QueryGeometry();

/*
 * Private Function Definitions.
 */

static void GetDefaultSize(), DrawBitmaps(), GetBitmapInfo();
static void CreateGCs(), DestroyGCs();

#define superclass (&smeClassRec)
SmeBSBprClassRec smeBSBprClassRec = {

```

- 323 -

```
{
/* superclass      */ (WidgetClass) superclass,
/* class_name      */ "SmeBSBpr",
/* size            */ sizeof(SmeBSBprRec),
/* class_initializer */ ClassInitialize,
/* class_part_initialize*/ NULL,
/* Class init'ed    */ FALSE,
/* initialize       */ Initialize,
/* initialize_hook   */ NULL,
/* realize          */ NULL,
/* actions          */ NULL,
/* num_actions       */ ZERO,
/* resources         */ resources,
/* resource_count    */ XtNumber(resources),
/* xrm_class         */ NULLQUARK,
/* compress_motion   */ FALSE,
/* compress_exposure */ FALSE,
/* compress_enterleave*/ FALSE,
/* visible_interest */ FALSE,
/* destroy           */ Destroy,
/* resize           */ NULL,
/* expose           */ Redisplay,
/* set_values        */ SetValues,
/* set_values_hook    */ NULL,
/* set_values_almost */ XtInheritSetValuesAlmost,
/* get_values_hook    */ NULL,
/* accept_focus       */ NULL,
/* intrinsics version */ XtVersion,
/* callback offsets   */ NULL,
/* tm_table           */ NULL,
/* query_geometry     */ QueryGeometry,
/* display_accelerator*/ NULL,
```

- 324 -

```

    /* extension      */ NULL
}, {
    /* Menu Entry Fields */

    /* highlight */      FlipColors,
    /* unhighlight */    FlipColors,
    /* notify */        PopupMenu,
    /* extension      */ NULL
}, {
    /* BSB pull-right Menu entry Fields */

    /* extension      */ NULL
}
};

WidgetClass smeBSBprObjectClass = (WidgetClass) &smeBSBprClassRec;

/*****
 *
 * Semi-Public Functions.
 *
 *****/

/* Function Name: ClassInitialize
 * Description: Initializes the SmeBSBprObject.
 * Arguments: none.
 * Returns: none.
 */

static void
ClassInitialize()
{

```

- 325 -

```
XawInitializeWidgetSet();
XtAddConverter( XtRString, XtRJustify, XmuCvtStringToJustify, NULL, 0 );
}
```

```
/* Function Name: Initialize
 * Description: Initializes the simple menu widget
 * Arguments: request - the widget requested by the argument list.
 *            new      - the new widget with both resource and non
 *                      resource values.
 * Returns: none.
 */
```

```
/* ARGSUSED */
```

```
static void
```

```
Initialize(request, new)
```

```
Widget request, new;
```

```
{
```

```
    SmeBSBprObject entry = (SmeBSBprObject) new;
```

```
    if (entry->sme_bsb.label == NULL)
```

```
        entry->sme_bsb.label = XtName(new);
```

```
    else
```

```
        entry->sme_bsb.label = XtNewString( entry->sme_bsb.label );
```

```
    /* Xaw bug - bitmap initialization now performed */
```

```
    if (entry->sme_bsb.left_bitmap != None) GetBitmapInfo(entry, TRUE);
```

```
    if (entry->sme_bsb.right_bitmap != None) GetBitmapInfo(entry, FALSE);
```

```
    CreateGCs(new);
```

```
    GetDefaultSize(new, &(entry->rectangle.width), &(entry->rectangle.height));
```

```
}
```

- 326 -

```
/*  Function Name: Destroy
 *  Description: Called at destroy time, cleans up.
 *  Arguments: w - the simple menu widget.
 *  Returns: none.
 */
```

```
static void
Destroy(w)
Widget w;
{
    SmeBSBprObject entry = (SmeBSBprObject) w;

    DestroyGCs(w);
    if (entry->sme_bsb.label != XtName(w))
        XtFree(entry->sme_bsb.label);
}
```

```
/*  Function Name: Redisplay
 *  Description: Redisplays the contents of the widget.
 *  Arguments: w - the simple menu widget.
 *              event - the X event that caused this redisplay.
 *              region - the region the needs to be repainted.
 *  Returns: none.
 */
```

```
/* ARGSUSED */
```

```
static void
Redisplay(w, event, region)
Widget w;
XEvent * event;
Region region;
{
```

- 327 -

GC gc;

SmeBSBprObject entry = (SmeBSBprObject) w;

int font_ascent, font_descent, y_loc;

entry->sme_bsb.set_values_area_cleared = FALSE;

font_ascent = entry->sme_bsb.font->max_bounds.ascent;

font_descent = entry->sme_bsb.font->max_bounds.descent;

y_loc = entry->rectangle.y;

if (XtIsSensitive(w) && XtIsSensitive(XtParent(w))) {

if (w == XawSimpleMenuGetActiveEntry(XtParent(w))) {

XFillRectangle(XtDisplayOfObject(w), XtWindowOfObject(w),

entry->sme_bsb.norm_gc, 0, y_loc,

(unsigned int) entry->rectangle.width,

(unsigned int) entry->rectangle.height);

gc = entry->sme_bsb.rev_gc;

}

else

gc = entry->sme_bsb.norm_gc;

}

else

gc = entry->sme_bsb.norm_gray_gc;

if (entry->sme_bsb.label != NULL) {

int x_loc = entry->sme_bsb.left_margin;

int len = strlen(entry->sme_bsb.label);

char * label = entry->sme_bsb.label;

switch(entry->sme_bsb.justify) {

int width, t_width;

- 328 -

```
case XtJustifyCenter:
```

```
    t_width = XTextWidth(entry->sme_bsb.font, label, len);
```

```
    width = entry->rectangle.width - (entry->sme_bsb.left_margin +
                                      entry->sme_bsb.right_margin);
```

```
    x_loc += (width - t_width)/2;
```

```
    break;
```

```
case XtJustifyRight:
```

```
    t_width = XTextWidth(entry->sme_bsb.font, label, len);
```

```
    x_loc = entry->rectangle.width - (entry->sme_bsb.right_margin +
                                      t_width);
```

```
    break;
```

```
case XtJustifyLeft:
```

```
default:
```

```
    break;
```

```
}
```

```
y_loc += (entry->rectangle.height -
          (font_ascent + font_descent)) / 2 + font_ascent;
```

```
XDrawString(XtDisplayOfObject(w), XtWindowOfObject(w), gc,
            x_loc, y_loc, label, len);
```

```
}
```

```
DrawBitmaps(w, gc);
```

```
}
```

```
/* Function Name: SetValues
```

```
* Description: Relayout the menu when one of the resources is changed.
```

```
* Arguments: current - current state of the widget.
```

```
*           request - what was requested.
```

```
*           new - what the widget will become.
```

- 329 -

* Returns: none

*/

/* ARGSUSED */

static Boolean

SetValues(current, request, new)

Widget current, request, new;

{

SmeBSBprObject entry = (SmeBSBprObject) new;

SmeBSBprObject old_entry = (SmeBSBprObject) current;

Boolean ret_val = FALSE;

if (old_entry->sme_bsb.label != entry->sme_bsb.label) {

if (old_entry->sme_bsb.label != XtName(new))

XtFree((char *) old_entry->sme_bsb.label);

if (entry->sme_bsb.label != XtName(new))

entry->sme_bsb.label = XtNewString(entry->sme_bsb.label);

ret_val = True;

}

if (entry->rectangle.sensitive != old_entry->rectangle.sensitive)

ret_val = TRUE;

if (entry->sme_bsb.left_bitmap != old_entry->sme_bsb.left_bitmap) {

GetBitmapInfo(new, TRUE);

ret_val = TRUE;

}

if (entry->sme_bsb.right_bitmap != old_entry->sme_bsb.right_bitmap) {

GetBitmapInfo(new, FALSE);

- 330 -

```

    ret_val = TRUE;
}

if ( (old_entry->sme_bsb.font != entry->sme_bsb.font) ||
      (old_entry->sme_bsb.foreground != entry->sme_bsb.foreground) ) {
    DestroyGCs(current);
    CreateGCs(new);
    ret_val = TRUE;
}

if (ret_val) {
    GetDefaultSize(new,
                    &(entry->rectangle.width), &(entry->rectangle.height));
    entry->sme_bsb.set_values_area_cleared = TRUE;
}
return(ret_val);
}

/*  Function Name: QueryGeometry.
 *   Description: Returns the preferred geometry for this widget.
 *   Arguments: w - the menu entry object.
 *              intended, return_val - the intended and return geometry info.
 *   Returns: A Geometry Result.
 *
 *   See the Intrinsics manual for details on what this function is for.
 *
 *   I just return the height and width of the label plus the margins.
 */

```

```

static XtGeometryResult
QueryGeometry(w, intended, return_val)
Widget w;

```

- 331 -

```
XtWidgetGeometry *intended, *return_val;
{
    SmeBSBprObject entry = (SmeBSBprObject) w;
    Dimension width, height;
    XtGeometryResult ret_val = XtGeometryYes;
    XtGeometryMask mode = intended->request_mode;

    GetDefaultSize(w, &width, &height);

    if ( ((mode & CWWidth) && (intended->width != width)) ||
        !(mode & CWWidth) ) {
        return_val->request_mode |= CWWidth;
        return_val->width = width;
        ret_val = XtGeometryAlmost;
    }

    if ( ((mode & CWHeight) && (intended->height != height)) ||
        !(mode & CWHeight) ) {
        return_val->request_mode |= CWHeight;
        return_val->height = height;
        ret_val = XtGeometryAlmost;
    }

    if (ret_val == XtGeometryAlmost) {
        mode = return_val->request_mode;

        if ( ((mode & CWWidth) && (width == entry->rectangle.width)) &&
            ((mode & CWHeight) && (height == entry->rectangle.height)) )
            return(XtGeometryNo);
    }

    return(ret_val);
}
```

- 332 -

}

```

/*    Function Name: FlipColors
 *    Description: Invert the colors of the current entry.
 *    Arguments: w - the bsb menu entry widget.
 *    Returns: none.
 */

```

static void

FlipColors(w)

Widget w;

{

```

    SmeBSBprObject entry = (SmeBSBprObject) w;

```

```

    if (entry->sme_bsb.set_values_area_cleared) return;

```

```

    XFillRectangle(XtDisplayOfObject(w), XtWindowOfObject(w),
        entry->sme_bsb.invert_gc, 0, (int) entry->rectangle.y,
        (unsigned int) entry->rectangle.width,
        (unsigned int) entry->rectangle.height);

```

}

```

/*****

```

*

* Private Functions.

*

```

*****/

```

```

/*    Function Name: GetDefaultSize
 *    Description: Calculates the Default (preferred) size of
 *                  this menu entry.
 *    Arguments: w - the menu entry widget.

```

- 333 -

```
*      width, height - default sizes (RETURNED).
*
*      Returns: none.
*/
```

```
static void
```

```
GetDefaultSize(w, width, height)
```

```
Widget w;
```

```
Dimension * width, * height;
```

```
{
```

```
    SmeBSBprObject entry = (SmeBSBprObject) w;
```

```
    if (entry->sme_bsb.label == NULL)
```

```
        *width = 0;
```

```
    else
```

```
        *width = XTextWidth(entry->sme_bsb.font, entry->sme_bsb.label,
                             strlen(entry->sme_bsb.label));
```

```
    *width += entry->sme_bsb.left_margin + entry->sme_bsb.right_margin;
```

```
    *height = (entry->sme_bsb.font->max_bounds.ascent +
               entry->sme_bsb.font->max_bounds.descent);
```

```
    *height = (*height * ( ONE_HUNDRED +
                           entry->sme_bsb.vert_space )) / ONE_HUNDRED;
```

```
}
```

```
/*      Function Name: DrawBitmaps
```

```
*      Description: Draws left and right bitmaps.
```

```
*      Arguments: w - the simple menu widget.
```

```
*      gc - graphics context to use for drawing.
```

```
*      Returns: none
```

```
*/
```

- 334 -

```
static void
DrawBitmaps(w, gc)
Widget w;
GC gc;
{
    int x_loc, y_loc;
    SmeBSBprObject entry = (SmeBSBprObject) w;

    if ( (entry->sme_bsb.left_bitmap == None) &&
        (entry->sme_bsb.right_bitmap == None) ) return;

    /*
     * Draw Left Bitmap.
     */

    y_loc = entry->rectangle.y + (entry->rectangle.height -
                                   entry->sme_bsb.left_bitmap_height) / 2;

    if (entry->sme_bsb.left_bitmap != None) {
        x_loc = (entry->sme_bsb.left_margin -
                 entry->sme_bsb.left_bitmap_width) / 2;
        XCopyPlane(XtDisplayOfObject(w), entry->sme_bsb.left_bitmap,
                   XtWindowOfObject(w), gc, 0, 0,
                   entry->sme_bsb.left_bitmap_width,
                   entry->sme_bsb.left_bitmap_height, x_loc, y_loc, 1);
    }

    /*
     * Draw Right Bitmap.
     */

    y_loc = entry->rectangle.y + (entry->rectangle.height - /* Xaw bug - y_loc
```

- 335 -

calculated from right_bitmap data */

entry->sme_bsb.right_bitmap_height) / 2;

if (entry->sme_bsb.right_bitmap != None) {

x_loc = entry->rectangle.width - (entry->sme_bsb.right_margin + /* Xaw bug - +
rather than - sign */

entry->sme_bsb.right_bitmap_width) / 2;

XCopyPlane(XtDisplayOfObject(w), entry->sme_bsb.right_bitmap,

XtWindowOfObject(w), gc, 0, 0,

entry->sme_bsb.right_bitmap_width,

entry->sme_bsb.right_bitmap_height, x_loc, y_loc, 1);

}

}

/* Function Name: GetBitmapInfo

* Description: Gets the bitmap information from either of the bitmaps.

* Arguments: w - the bsb menu entry widget.

* is_left - TRUE if we are testing left bitmap,

* FALSE if we are testing the right bitmap.

* Returns: none

*/

static void

GetBitmapInfo(w, is_left)

Widget w;

Boolean is_left;

{

SmeBSBprObject entry = (SmeBSBprObject) w;

unsigned int depth, bw;

Window root;

int x, y;

unsigned int width, height;

- 336 -

```
char buf[BUFSIZ];
```

```
if (is_left) {
```

```
    if (entry->sme_bsb.left_bitmap != None) {
```

```
        if (!XGetGeometry(XtDisplayOfObject(w),
```

```
            entry->sme_bsb.left_bitmap, &root,
```

```
            &x, &y, &width, &height, &bw, &depth)) {
```

```
            sprintf(buf, "SmeBSB Object: %s %s \"%s\".", "Could not",
```

```
                "get Left Bitmap geometry information for menu entry ",
```

```
                XtName(w));
```

```
            XtAppError(XtWidgetToApplicationContext(w), buf);
```

```
        }
```

```
        if (depth != 1) {
```

```
            sprintf(buf, "SmeBSB Object: %s \"%s\"%s.",
```

```
                "Left Bitmap of entry ",
```

```
                XtName(w), " is not one bit deep.");
```

```
            XtAppError(XtWidgetToApplicationContext(w), buf);
```

```
        }
```

```
        entry->sme_bsb.left_bitmap_width = (Dimension) width;
```

```
        entry->sme_bsb.left_bitmap_height = (Dimension) height;
```

```
    }
```

```
}
```

```
else if (entry->sme_bsb.right_bitmap != None) {
```

```
    if (!XGetGeometry(XtDisplayOfObject(w),
```

```
        entry->sme_bsb.right_bitmap, &root,
```

```
        &x, &y, &width, &height, &bw, &depth)) {
```

```
        sprintf(buf, "SmeBSB Object: %s %s \"%s\".", "Could not",
```

```
            "get Right Bitmap geometry information for menu entry ",
```

```
            XtName(w));
```

```
        XtAppError(XtWidgetToApplicationContext(w), buf);
```

```
    }
```

```
    if (depth != 1) {
```

- 337 -

```
    sprintf(buf, "SmeBSB Object: %s \"%s\" \"%s\".",
```

```
        "Right Bitmap of entry ", XtName(w),
```

```
        " is not one bit deep.");
```

```
    XtAppError(XtWidgetToApplicationContext(w), buf);
```

```
    }
```

```
    entry->sme_bsb.right_bitmap_width = (Dimension) width;
```

```
    entry->sme_bsb.right_bitmap_height = (Dimension) height;
```

```
    }
```

```
}
```

```
/*  Function Name: CreateGCs
```

```
    *  Description: Creates all gc's for the simple menu widget.
```

```
    *  Arguments: w - the simple menu widget.
```

```
    *  Returns: none.
```

```
*/
```

```
static void
```

```
CreateGCs(w)
```

```
Widget w;
```

```
{
```

```
    SmeBSBprObject entry = (SmeBSBprObject) w;
```

```
    XGCValues values;
```

```
    XtGCMask mask;
```

```
    values.foreground = XtParent(w)->core.background_pixel;
```

```
    values.background = entry->sme_bsb.foreground;
```

```
    values.font = entry->sme_bsb.font->fid;
```

```
    values.graphics_exposures = FALSE;
```

```
    mask      = GCForeground | GCBackground | GCFont | GCGraphicsExposures;
```

```
    entry->sme_bsb.rev_gc = XtGetGC(w, mask, &values);
```

```
    values.foreground = entry->sme_bsb.foreground;
```

- 338 -

```

values.background = XtParent(w)->core.background_pixel;
entry->sme_bsb.norm_gc = XtGetGC(w, mask, &values);

values.fill_style = FillTiled;
values.tile = XmuCreateStippledPixmap(XtScreenOfObject(w),
                                     entry->sme_bsb.foreground,
                                     XtParent(w)->core.background_pixel,
                                     XtParent(w)->core.depth);

values.graphics_exposures = FALSE;
mask |= GCTile | GCFillStyle;
entry->sme_bsb.norm_gray_gc = XtGetGC(w, mask, &values);

values.foreground ^= values.background;
values.background = 0;
values.function = GXxor;
mask = GCForeground | GCBackground | GCGraphicsExposures | GCFunction;
entry->sme_bsb.invert_gc = XtGetGC(w, mask, &values);
}

/*  Function Name: DestroyGCs
 *   Description: Removes all gc's for the simple menu widget.
 *   Arguments: w - the simple menu widget.
 *   Returns: none.
 */

static void
DestroyGCs(w)
Widget w;
{
    SmeBSBprObject entry = (SmeBSBprObject) w;

    XtReleaseGC(w, entry->sme_bsb.norm_gc);

```

- 339 -

```
    XtReleaseGC(w, entry->sme_bsb.norm_gray_gc);
    XtReleaseGC(w, entry->sme_bsb.rev_gc);
    XtReleaseGC(w, entry->sme_bsb.invert_gc);
}

#ifdef apollo

/*
 * The apollo compiler that we have optimizes out my code for
 * FlipColors() since it is static. and no one executes it in this
 * file. I am setting the function pointer into the class structure so
 * that it can be called by my parent who will tell me to when to
 * highlight and unhighlight.
 */

void _XawSmeBSBApolloHack ()
{
    FlipColors();
}

#endif /* apollo */

/* Hacked copy of PopupMenu from MenuButton widget to replace XUnheritNotify */

static void
PopupMenu(w, event, params, num_params)
Widget w;
XEvent * event;
String * params;
Cardinal * num_params;
{
    SmeBSBprObject mbw = (SmeBSBprObject) w;
    Widget menu, temp;
```

- 340 -

```
Arg arglist[2];
Cardinal num_args;
int menu_x, menu_y, menu_width, menu_height, button_width, button_height;
Position button_x, button_y;

temp = XtParent(w); /* Shell not menu entry is parent of menu */
while(temp != NULL) {
    menu = XtNameToWidget(temp, mbw->sme_bsb.menu_name);
    if (menu == NULL)
        temp = XtParent(temp);
    else
        break;
}

if (menu == NULL) {
    char error_buf[BUFSIZ];
    sprintf(error_buf, "MenuButton: %s %s.",
            "Could not find menu widget named", mbw->sme_bsb.menu_name);
    XtAppWarning(XtWidgetToApplicationContext(w), error_buf);
    return;
}
if (!XtIsRealized(menu))
    XtRealizeWidget(menu);

menu_width = menu->core.width + 2 * menu->core.border_width;
button_width = w->core.width + 2 * w->core.border_width;
button_height = w->core.height + 2 * w->core.border_width;

menu_height = menu->core.height + 2 * menu->core.border_width;

XtTranslateCoords(w, 0, 0, &button_x, &button_y);
menu_x = button_x + button_width;
```

- 341 -

```
menu_y = button_y;
```

```
if (menu_x < 0)
```

```
    menu_x = 0;
```

```
else {
```

```
    int scr_width = WidthOfScreen(XtScreen(menu));
```

```
    if (menu_x + menu_width > scr_width)
```

```
        menu_x = scr_width - menu_width;
```

```
}
```

```
if (menu_y < 0)
```

```
    menu_y = 0;
```

```
else {
```

```
    int scr_height = HeightOfScreen(XtScreen(menu));
```

```
    if (menu_y + menu_height > scr_height)
```

```
        menu_y = scr_height - menu_height;
```

```
}
```

```
num_args = 0;
```

```
XtSetArg(arglist[num_args], XtNx, menu_x); num_args++;
```

```
XtSetArg(arglist[num_args], XtNy, menu_y); num_args++;
```

```
XtSetValues(menu, arglist, num_args);
```

```
XtPopupSpringLoaded(menu);
```

```
}
```

- 342 -

source/Storage.c

/*

Routines to allow video frames to be stored in memory

or on disk: NewFrame, GetFrame, SaveFrame, FreeFrame, SaveHeader,

CopyHeader.

*/

#include "../include/xwave.h"

extern FILE *zopen();

extern void zseek();

extern void zclose();

void NewFrame(vid,number)

Video vid;

int number;

{

if (vid->data[0][number] == NULL) {

int channel, channels = vid->type == MONO?1:3;

for(channel=0;channel < channels;channel++)

vid->data[channel][number] = (short

*)MALLOC(sizeof(short)*Size(vid,channel,0)*Size(vid,channel,1));

}

}

void GetFrame(vid,number)

Video vid;

- 343 -

```

int    number;

{
    if (vid->data[0][number] == NULL) {
        char    file_name[STRLEN], *whole_frame;
        FILE    *fp, *fopen();
        int     pid, r, c, channel,
                start = vid->x_offset + vid->cols*vid->y_offset,

end = (vid->rows-vid->y_offset-vid->size[1])*vid->cols-vid->x_offset,
        inter = vid->cols-vid->size[0];

        NewFrame(vid,number);

        sprintf(file_name, "%s%s/%s/%s%03d\0", global->home, IMAGE_DIR, vid->path, vid->files[0] = '\0'?vid->name:vid->files,number+vid->start);
        Dprintf("Reading file %s\n", file_name);
        fp = zreopen(file_name, &pid);
        if (vid->precision == 0) whole_frame = (char
*)MALLOC(vid->rows*vid->cols);
        zseek(fp, vid->offset);
        for(channel=0; channel < (vid->type == MONO?1:3); channel++) {
            int    shift[2] = {vid->type == YUV &&
channel != 0?vid->UVsample[0]:0, vid->type == YUV &&
channel != 0?vid->UVsample[1]:0};

            Dprintf("Reading channel %d\n", channel);
            if (vid->precision == 0) {

if(0 == fread(whole_frame, sizeof(char), (vid->cols >> shift[0])*(vid->rows >> shift[1]),
fp)) {
                Dprintf("Error whilst reading %s\n", file_name);

```


- 344 -

```

        Eprintf("Error whilst reading %s\n",file_name);
    }
    for(r=0;r<vid->size[1]>>shift[1];r++)
        for(c=0;c<vid->size[0]>>shift[0];c++) {
            short
            pel=cti(whole_frame[(vid->x_offset>>shift[0])+c+((vid->y_offset>>shift[1])+r)*
            vid->cols>>shift[0]]);

            vid->data[channel][number][c+r*(vid->size[0]>>shift[0])]=vid->negative?-1-pel:pel;
        }
    } else {
        if (start!=0) zseek(fp,start*sizeof(short));
        for(r=0;r<vid->size[1]>>shift[1];r++) {

            if(0==fread(&(vid->data[channel][number][r*(vid->size[0]>>shift[0])]),sizeof(short),
            vid->size[0]>>shift[0],fp)) {

                Dprintf("Error whilst reading

                %s\n",file_name);

                Eprintf("Error whilst reading

                %s\n",file_name);
            }
            if (inter!=0) zseek(fp,inter*sizeof(short));
            if (vid->negative)
                for(c=0;c<vid->size[0]>>shift[0];c++)

            vid->data[channel][number][c+r*(vid->size[0]>>shift[0])]=-1-vid->data[channel][nu
            mber][c+r*(vid->size[0]>>shift[0])];

```

- 345 -

source/Storage.c

```
/*  
    Routines to allow video frames to be stored in memory  
    or on disk: NewFrame, GetFrame, SaveFrame, FreeFrame, SaveHeader,  
    CopyHeader.  
*/
```

```
#include    "../include/xwave.h"
```

```
extern FILE *zopen();
```

```
extern void zseek();
```

```
extern void zclose();
```

```
void NewFrame(vid,number)
```

```
Video vid;
```

```
int number;
```

```
{  
    if (vid->data[0][number] == NULL) {  
        int channel, channels=vid->type == MONO?1:3;  
  
        for(channel=0;channel < channels;channel++)  
            vid->data[channel][number] = (short  
*)MALLOC(sizeof(short)*Size(vid,channel,0)*Size(vid,channel,1));  
    }  
}
```

```
void GetFrame(vid,number)
```

```
Video vid;
```

- 346 -

```

int    number;

{
    if (vid->data[0][number] == NULL) {
        char    file_name[STRLEN], *whole_frame;
        FILE    *fp, *fopen();
        int      pid, r, c, channel,
                start=vid->x_offset+vid->cols*vid->y_offset,

end=(vid->rows-vid->y_offset-vid->size[1])*vid->cols-vid->x_offset,
        inter=vid->cols-vid->size[0];

        NewFrame(vid,number);

        sprintf(file_name,"%s%s/%s/%s%03d\0",global->home,IMAGE_DIR,vid->path,vid->files[0] == '\0'?vid->name:vid->files,number+vid->start);
        Dprintf("Reading file %s\n",file_name);
        fp=zopen(file_name,&pid);
        if (vid->precision==0) whole_frame=(char
*)MALLOC(vid->rows*vid->cols);
        zseek(fp,vid->offset);
        for(channel=0;channel<(vid->type==MONO?1:3);channel++) {
            int    shift[2]={vid->type==YUV &&
channel!=0?vid->UVsample[0]:0,vid->type==YUV &&
channel!=0?vid->UVsample[1]:0};

            Dprintf("Reading channel %d\n",channel);
            if (vid->precision==0) {

if(0==fread(whole_frame,sizeof(char),(vid->cols>>shift[0])*(vid->rows>>shift[1]),
fp)) {

                Dprintf("Error whilst reading %s\n",file_name);

```

- 347 -

```

        Eprintf("Error whilst reading %s\n",file_name);
    }
    for(r=0;r < vid-> size[1] > > shift[1];r++)
        for(c=0;c < vid-> size[0] > > shift[0];c++) {
            short
            pel=cti(whole_frame[(vid-> x_offset > > shift[0])+c+((vid-> y_offset > > shift[1])+r)*(
            vid-> cols > > shift[0])));

            vid-> data[channel][number][c+r*(vid-> size[0] > > shift[0])] = vid-> negative?-1-pel:pel;
        }
    } else {
        if (start!=0) zseek(fp,start*sizeof(short));
        for(r=0;r < vid-> size[1] > > shift[1];r++) {

            if(0==fread(&(vid-> data[channel][number][r*(vid-> size[0] > > shift[0]))),sizeof(short),
            vid-> size[0] > > shift[0],fp)) {

                Dprintf("Error whilst reading

                %s\n",file_name);

                Eprintf("Error whilst reading

                %s\n",file_name);

            }
            if (inter!=0) zseek(fp,inter*sizeof(short));
            if (vid-> negative)
                for(c=0;c < vid-> size[0] > > shift[0];c++)

            vid-> data[channel][number][c+r*(vid-> size[0] > > shift[0])] = -1-vid-> data[channel][nu
            mber][c+r*(vid-> size[0] > > shift[0])];

```

- 348 -

}

```
void SaveHeader(vid)
```

```
Video vid;
```

{

```
FILE *fp, *fopen();
```

```
char file_name[STRLEN];
```

```
String types[] = {"MONO", "RGB", "YUV"};
```

```
Dprintf("SaveHeader %s\n", vid->name);
```

```
sprintf(file_name, "%s%s/%s%s\0", global->home, VID_DIR, vid->name, VID_EXT);
```

```
fp = fopen(file_name, "w");
```

```
fprintf(fp, "Path \" %s\" \n", vid->path);
```

```
if (vid->files[0] != '\0') fprintf(fp, "Files \" %s\" \n", vid->files);
```

```
if (vid->type == YUV) fprintf(fp, "Type %s %d
```

```
%d\n", types[vid->type], vid->UVsample[0], vid->UVsample[1]);
```

```
else fprintf(fp, "Type %s\n", types[vid->type]);
```

```
if (vid->rate != 0) fprintf(fp, "Rate %d\n", vid->rate);
```

```
if (vid->disk) fprintf(fp, "Disk\n");
```

```
if (vid->gamma) fprintf(fp, "Gamma\n");
```

```
fprintf(fp, "Start %03d\n", vid->start);
```

```
fprintf(fp, "Length %d\n", vid->size[2]);
```

```
fprintf(fp, "Dimensions %d %d\n", vid->cols, vid->rows);
```

```
switch(vid->trans.type) {
```

```
case TRANS_None: fprintf(fp, "Transform None\n"); break;
```

```
case TRANS_Wave: fprintf(fp, "Transform Wavelet %d %d
```

```
%s\n", vid->trans.wavelet.space[0], vid->trans.wavelet.space[1], vid->trans.wavelet.dirn
```

```
?"Yes":"No"); break;
```

- 349 -

```

    }
    fprintf(fp, "Header %d\n", vid->offset);
    fprintf(fp, "Offsets %d %d\n", vid->x_offset, vid->y_offset);
    fprintf(fp, "Size %d %d\n", vid->size[0], vid->size[1]);
    fprintf(fp, "Precision %d\n", vid->precision);
    fclose(fp);
}

```

Video CopyHeader(src)

Video src;

```

{
    Video dst=(Video)MALLOC(sizeof(VideoRec));
    int channel;

    Dprintf("CopyHeader %s\n",src);
    strcpy(dst->path,src->path);
    strcpy(dst->name,src->name);
    dst->type=src->type;
    dst->disk=src->disk;
    dst->gamma=src->gamma;
    dst->negative=False;
    dst->rate=src->rate;
    dst->start=src->start;
    dst->size[0]=src->size[0];
    dst->size[1]=src->size[1];
    dst->size[2]=src->size[2];
    dst->UVsample[0]=src->UVsample[0];
    dst->UVsample[1]=src->UVsample[1];
    dst->offset=0;
    dst->cols=src->size[0];
}

```

- 350 -

```
dst->rows=src->size[1];
dst->x_offset=0;
dst->y_offset=0;
dst->trans=src->trans;
dst->precision=src->precision;
for(channel=0;channel<(src->type==MONO?1:3):channel++)
    dst->data[channel]=(short **)MALLOC(src->size[2]*sizeof(short *));
return(dst);
```

}

- 351 -

source/Transform.c

```
/*
    Transform video using wavelet transform
*/

#include "xwave.h"
#include "Transform.h"
extern short Round();

void DropVideo(w, closure, call_data)

Widget      w;
caddr_t     closure, call_data;

{
    Video video = global->videos->next;
    int     frame, channel;

    for(channel=0; channel < (global->videos->type == MONO?1:(global->videos->type == YUV?3:4)); channel++)
        if (global->videos->data[channel] != NULL) {
            for (frame=0; frame < global->videos->size[2]; frame++)
                if (global->videos->data[channel][frame] != NULL)
                    XtFree(global->videos->data[channel][frame]);
            XtFree(global->videos->data[channel]);
        }

    XtFree(global->videos);
    global->videos = video;
}
```


- 352 -

}

```
void ChangePrecision(src,dst,frame,old,new)
```

```
Video src, dst;
```

```
int frame, old, new;
```

{

```
int channel, i;
```

```
if(src!=dst || old!=new) {
```

```
int shift=new-old;
```

```
Dprintf("Changing precision %d to %d for frame %d\n",old,new,frame);
```

```
for (channel=0;channel<(src->type==MONO?1:3);channel++) {
```

```
int size=Size(src,channel,0)*Size(src,channel,1);
```

```
for(i=0;i<size;i++)
```

```
dst->data[channel][frame][i]=shift<0?Round(src->data[channel][frame][i],-shift):(shift
```

```
==0?src->data[channel][frame][i]:src->data[channel][frame][i]<<shift);
```

```
}
```

```
}
```

}

```
void TransformCtrl(w,closure,call_data)
```

```
Widget w;
```

```
caddr_t closure, call_data;
```

{

```
TransCtrl ctrl=(TransCtrl)closure;
```

- 353 -

```

Video src=ctrl->src, dst=CopyHeader(src);
long i, frame, channel;

Dprintf("TransformCtrl\n");
strcpy(dst->name,ctrl->name);
dst->trans.type=TRANS_Wave;
dst->trans.wavelet.space[0]=ctrl->space[0];
dst->trans.wavelet.space[1]=ctrl->space[1];
dst->trans.wavelet.dirn=ctrl->dirn;
dst->precision=ctrl->precision;
strcpy(dst->files,dst->name);
if (dst->disk) SaveHeader(dst);
if (src->trans.type!=TRANS_Wave) {
    src->trans.type=TRANS_Wave;
    src->trans.wavelet.space[0]=0;
    src->trans.wavelet.space[1]=0;
}

if (src->trans.wavelet.space[0]!=dst->trans.wavelet.space[0] ||
src->trans.wavelet.space[1]!=dst->trans.wavelet.space[1])
    for(frame=0;frame<dst->size[2];frame++) {
        int
max_precision=src->precision>dst->precision?src->precision:dst->precision;

        Dprintf("Processing frame %d\n",frame);
        NewFrame(dst,frame);
        GetFrame(src,frame);
        ChangePrecision(src,dst,frame,src->precision,max_precision);
        for (channel=0;channel<(src->type==MONO?1:3);channel++)
        {
            int oct_src=src->trans.wavelet.space[channel==0?0:1],

```

- 354 -

```
oct_dst = dst->trans.wavelet.space[channel == 0?0:1],
```

```
size[2] = {Size(dst,channel,0),Size(dst,channel,1)};
```

```
if (oct_src != oct_dst)
```

```
Convoive(dst->data[channel][frame],curl->dim.size,oct_src,oct_dst);
```

```
}
```

```
ChangePrecision(dst,dst,frame,max_precision,dst->precision);
```

```
SaveFrame(dst,frame);
```

```
FreeFrame(dst,frame);
```

```
FreeFrame(src,frame);
```

```
}
```

```
if (src->trans.wavelet.space[0] == 0 && src->trans.wavelet.space[1] == 0)
```

```
src->trans.type = TRANS_None;
```

```
if (dst->trans.wavelet.space[0] == 0 && dst->trans.wavelet.space[1] == 0) {
```

```
dst->trans.type = TRANS_None;
```

```
if (dst->disk) SaveHeader(dst);
```

```
}
```

```
dst->next = global->videos;
```

```
global->videos = dst;
```

```
}
```

```
void Transtype(w,closure,call_data)
```

```
Widget w;
```

```
caddr_t closure, call_data;
```

```
{
```

```
Video vid = (Video)closure;
```

```
if (vid->trans.wavelet.space[0] == 0 && vid->trans.wavelet.space[1] == 0)
```

- 355 -

```
vid->trans.type = TRANS_None;
}
```

```
void BatchTransCtrl(w,closure,call_data)
```

```
Widget      w;
```

```
caddr_t     closure, call_data;
```

```
{
    TransCtrl ctrl=(TransCtrl)closure;

    if (ctrl->src == NULL) ctrl->src = FindVideo(ctrl->src_name,global->videos);
    if (ctrl->src->trans.type == TRANS_Wave)
ctrl->dirn = ctrl->src->trans.wavelet.dirn;
    TransformCtrl(w,closure,call_data);
}
```

```
TransCtrl InitTransCtrl(name)
```

```
String name;
```

```
{
    TransCtrl ctrl=(TransCtrl)MALLOC(sizeof(TransCtrlRec));

    strcpy(ctrl->src_name,name);
    strcpy(ctrl->name,name);
    ctrl->dirn=False;
    Dprintf("Transform\n");
    return(ctrl);
}
```

```
#define TRANS_ICONS 16
```

- 356 -

```
void Transform(w,closure,call_data)
```

```
Widget      w;
```

```
caddr_t      closure, call_data;
```

```
{
```

```
    Video video=(Video)closure;
```

```
    TransCtrl ctrl=InitTransCtrl(video->name);
```

```
    NumInput spaceInput=(NumInput)MALLOC(2*sizeof(NumInputRec)),
```

```
                precInput=(NumInput)MALLOC(sizeof(NumInputRec));
```

```
    Message msg=NewMessage(ctrl->name,NAME_LEN);
```

```
    XtCallbackRec destroy_call[]={
```

```
        {Free,(caddr_t)ctrl},
```

```
        {Free,(caddr_t)spaceInput},
```

```
        {Free,(caddr_t)precInput},
```

```
        {CloseMessage,(caddr_t)msg},
```

```
        {NULL,NULL},
```

```
    };
```

```
    Widget parent=FindWidget("frm_transform",XtParent(w)),
```

```
    shell=ShellWidget("transform",parent,SW_below,NULL,destroy_call),
```

```
        form=FormatWidget("trans_form",shell),
```

```
    widgets[TRANS_ICONS];
```

```
    FormItem items[]={
```

```
        {"trans_cancel","cancel",0,0,FW_icon,NULL},
```

```
        {"trans_confirm","confirm",1,0,FW_icon,NULL},
```

```
        {"trans_title","Transform a video",2,0,FW_label,NULL},
```

```
        {"trans_vid_lab","Video Name:",0,3,FW_label,NULL},
```

```
        {"trans_video",NULL,4,3,FW_text,(String)msg},
```

```
        {"trans_dirn_lab","Direction:",0,4,FW_label,NULL},
```

```
        {"trans_dirn",NULL,4,4,FW_yn,(String)&ctrl->dirn},
```

- 357 -

```

    {"trans_bits_int", NULL, 0, 6, FW_integer, (String)precInput},
    {"trans_bits_down", NULL, 4, 6, FW_down, (String)precInput},
    {"trans_bits_up", NULL, 9, 6, FW_up, (String)precInput},

    {"trans_spc0_int", NULL, 0, 8, FW_integer, (String)&spaceInput[0]},
    {"trans_spc0_down", NULL, 4, 8, FW_down, (String)&spaceInput[0]},
    {"trans_spc0_up", NULL, 12, 8, FW_up, (String)&spaceInput[0]},
    {"trans_spc1_int", NULL, 0, 11, FW_integer, (String)&spaceInput[1]},
    {"trans_spc1_down", NULL, 4, 11, FW_down, (String)&spaceInput[1]},

    {"trans_spc1_up", NULL, 15, 11, FW_up, (String)&spaceInput[1]},
};

XtCallbackRec    callbacks[] = {
    {Destroy, (caddr_t)shell},
    {NULL, NULL},
    {TransformCtrl, (caddr_t)ctrl},
    {Destroy, (caddr_t)shell},
    {NULL, NULL},
    {ChangeYN, (caddr_t)&ctrl->dim}, {NULL, NULL},
    {NumIncDec, (caddr_t)precInput}, {NULL, NULL},
    {NumIncDec, (caddr_t)precInput}, {NULL, NULL},
    {NumIncDec, (caddr_t)&spaceInput[0]}, {NULL, NULL},
    {NumIncDec, (caddr_t)&spaceInput[0]}, {NULL, NULL},
    {NumIncDec, (caddr_t)&spaceInput[1]}, {NULL, NULL},
    {NumIncDec, (caddr_t)&spaceInput[1]}, {NULL, NULL},
};

Dprintf("Transform\n");
msg->rows = 1; msg->cols = NAME_LEN;
ctrl->src = video;
if (video->trans.type == TRANS_Wave) {
    ctrl->space[0] = video->trans.wavelet.space[0];

```

- 358 -

```
ctrl->space[1]=video->trans.wavelet.space[1];
ctrl->dim=video->trans.wavelet.dim;
} else {
    ctrl->space[0]=0; ctrl->space[1]=0;
    ctrl->dim=False;
}
ctrl->precision=video->precision;

spaceInput[0].format=video->type==YUV?"Y-Space: %d":"Space: %d";
spaceInput[0].max=100;
spaceInput[0].min=0;
spaceInput[0].value=&ctrl->space[0];
if (video->type==YUV) {
    spaceInput[1].format="UV-Space: %d";
    spaceInput[1].max=100;
    spaceInput[1].min=0;
    spaceInput[1].value=&ctrl->space[1];
}
preInput->format="Precision: %d";
preInput->max=16;
preInput->min=0;
preInput->value=&ctrl->precision;

FillForm(form,TRANS_ICONS-(video->type==YUV?0:3),items,widgets,callbacks);
if (video->trans.type==TRANS_Wave) XtSetSensitive(widgets[6],False);
XtPopup(shell,XtGrabExclusive);
}
```

- 359 -

source/Update.c

```
/*
    Update Image, Info and InfoText from positional information
*/

#include    "../include/xwave.h"
#include    <varargs.h>
extern int  CompositePixel();
extern int  Dither();
extern short Round();
extern int  ReMap();
extern Palette FindPalette();

char *ResizeData(size)

int  size;

{
    static char  *data=NULL;
    static int   data_size=0;

    if (size!=data_size) {
        Dprintf("New frame memory\n");
        if (data!=NULL) XtFree(data);
        data=(char *)MALLOC(size);
        data_size=size;
    }
    return(data);
}
```


- 360 -

Pixmap UpdateImage(frame)

Frame frame;

```
{
    int    x, y, i;
    Display      *dpy = XtDisplay(global->toplevel);
    void    CvtIndex(), UpdatePoint();
    Palette      pal = FindPalene(global->palettes, frame->palette);
    Video vid = frame->video;
    int    scrn = XDefaultScreen(dpy), depth = DisplayPlanes(dpy, scrn),
           size[2] = {Size(vid, frame->channel, 0), Size(vid, frame->channel, 1)},
           img_size[2] = {size[0] << frame->zoom, size[1] << frame->zoom},
           bpl = (img_size[0]*depth+7)/8, new_size = img_size[1]*bpl,
           space = vid->trans.wavelet.space[vid->type == YUV &&
frame->channel != 0 && frame->channel != 3?1:0];
    char    *data = ResizeData(new_size);
    XImage
    *image = XCreateImage(dpy, global->visinfo->visual, depth, ZPixmap, 0, data, img_size[0], i
mg_size[1], 8, bpl);
    Pixmap
    pixmap = XCreatePixmap(dpy, DefaultRootWindow(dpy), img_size[0], img_size[1], depth);

    Dprintf("UpdateImage\n");
    if (global->levels == 2 && frame->channel == 3) frame->channel = 0;
    for(y=0; y < size[1]; y++) for(x=0; x < size[0]; x++) {
        int    data_x = x, data_y = y, off_x, off_y, oct;

        if (vid->trans.type == TRANS_Wave)
            CvtIndex(x, y, size[0], size[1], space, &data_x, &data_y, &oct);
        for(off_x=0; off_x < 1 << frame->zoom; off_x++)
            for(off_y=0; off_y < 1 << frame->zoom; off_y++) {
```

- 361 -

```

        int    img_x=off_x+(x < < frame-> zoom),
img_y=off_y+(y < < frame-> zoom),

pix=CompositePixel(frame,data_x,data_y,img_x,img_y);

XPutPixel(image,img_x,img_y,ReMap(pix,global->levels,pal));
    }
}

XPutImage(dpy,pixmap,DefaultGC(dpy,scrn),image,0,0,0,0,img_size[0],img_size[1]);
    if (frame-> point_switch == True) UpdatePoint(dpy,frame,pixmap);
    XFree(image);
    return(pixmap);
}

void  CviIndex(x,y,max_x,max_y,oct,ret_x,ret_y,ret_oct)

int    x, y, max_x, max_y, oct, *ret_x, *ret_y, *ret_oct;

{
    Boolean    hgx=x>=(max_x>>1), hgy=y>=(max_y>>1);

    *ret_x=hgx?x-(max_x>>1):x;
    *ret_y=hgy?y-(max_y>>1):y;
    if (!hgx && !hgy && oct>1) {

CviIndex(*ret_x,*ret_y,max_x>>1,max_y>>1,oct-1,ret_x,ret_y,ret_oct);
        *ret_x= *ret_x<<1;
        *ret_y= *ret_y<<1;
        *ret_oct+=1;
    } else {

```

- 362 -

```
*ret_x=(*ret_x < 1)+hgx;
```

```
*ret_y=(*ret_y < 1)+hgy;
```

```
*ret_oct=hgx || hgy?0:1;
```

```
}
```

```
}
```

```
void UpdateInfo(frame)
```

```
Frame frame;
```

```
{
```

```
Message msg=frame->msg;
```

```
Video vid=frame->video;
```

```
int *locn=frame->point->location, posn[2]={locn[0],locn[1]},
```

```
channel=3==frame->channel?0:frame->channel,
```

```
width=Size(vid,channel,0);
```

```
short *data=vid->data[channel][frame->frame];
```

```
msg->info.ptr[0]='\0';
```

```
msg->info.length=0;
```

```
if (vid->type == YUV && channel!=0) {
```

```
posn[0]=posn[0]>>vid->UVsample[0];
```

```
posn[1]=posn[1]>>vid->UVsample[1];
```

```
}
```

```
if (vid->trans.type!=TRANS_Wave)
```

```
Mprintf(msg,"Point : x=%03d y=%03d t=%03d
```

```
c=%4d",locn[0],locn[1],frame->frame+vid->start,data[posn[0]+Size(vid,channel,0)*po  
sn[1]]);
```

```
else {
```

```
int octs=vid->trans.wavelet.space[vid->type == YUV &&
```

```
channel!=0?1:0],
```

```
X, Y, oct, sub,
```

- 363 -

```
blkDC[2] = {(posn[0] > > octs) & -2, (posn[1] > > octs) & -2},
```

```
offDC[2] = {(posn[0] > > octs) & 1, (posn[1] > > octs) & 1};
```

```
Mprintf(msg, "Point : f= %03d x= %03d
```

```
y= %03d\n", frame->frame + vid->start, locn[0], locn[1]);
```

```
Mprintf(msg, "Low pass: x= %03d y= %03d\n", blkDC[0], blkDC[1]);
```

```
for(Y=0; Y<2; Y++) {
```

```
    for(X=0; X<2; X++)
```

```
Mprintf(msg, " %4d%c", data[Access(blkDC[0]+X, blkDC[1]+Y, octs-1, 0, width)], X==off
DC[0] && Y==offDC[1]?'*':' ');
```

```
    Mprintf(msg, "\n");
```

```
}
```

```
for(oct=octs; oct>0; oct--) {
```

```
    int blk[2] = {(posn[0] > > oct) & -2, (posn[1] > > oct) & -2},
```

```
    off[2] = {(posn[0] > > oct) & 1, (posn[1] > > oct) & 1};
```

```
Mprintf(msg, "Oct : %d\n", oct);
```

```
for(Y=0; Y<2; Y++) {
```

```
    for(sub=1; sub<4; sub++) {
```

```
        for(X=0; X<2; X++) {
```

```
Mprintf(msg, " %4d%c", data[Access(blk[0]+X, blk[1]+Y, oct-1, sub, width)], X==off[0]
```

```
&& Y==off[1]?'*':' ');
```

```
        }
```

```
        if (sub<3) Mprintf(msg, " ");
```

```
    }
```

```
    if (oct!=0 || Y==0) Mprintf(msg, "\n");
```

```
}
```

```
}
```

```
}
```

- 364 -

Mflush(msg);

}

```

/*  Function Name:    CrossHair-
 *   Description:    Draws cross-hair on pixmap
 *   Arguments:     dpy - Xserver display
 *                  pixmap - pixmap to draw on
 *                  gc - GC to draw with
 *                  x_off, y_off - offset into pixmap
 *                  width, height - size of box containing cross-hair
 *                  x, y - coordinates within box
 *                  zoom - scaling factor
 *   Returns:       alters pixmap.
 */

```

```
void CrossHair(dpy,pixmap,gc,x_off,y_off,width,height,x,y,zoom)
```

```
Display    *dpy;
```

```
Pixmap     pixmap;
```

```
GC         gc;
```

```
int        x_off, y_off, width, height, x, y, zoom;
```

{

```
    int      xtra = Shift(1,zoom);
```

```
    x_off = Shift(x_off,zoom);
```

```
    y_off = Shift(y_off,zoom);
```

```
    width = Shift(width,zoom);
```

```
    height = Shift(height,zoom);
```

```
    x = Shift(x,zoom);
```

```
    y = Shift(y,zoom);
```

- 365 -

```

XFillRectangle(dpy,pixmap,gc,x+x_off+xtra/2,y_off,1,y); /* North hair */
XFillRectangle(dpy,pixmap,gc,x_off,y+y_off+xtra/2,x,1); /* West hair */
XFillRectangle(dpy,pixmap,gc,x+x_off+xtra/2,y+y_off+xtra,1,height-y-xtra); /*
South hair */
XFillRectangle(dpy,pixmap,gc,x+x_off+xtra,y+y_off+xtra/2,width-x-1,1); /*
East hair */
}

```

```

/*  Function Name:      UpdatePoint
*   Description:  Draws cross-hair on image at frame->location
*   Arguments:   dpy - X server display
*                frame - Frame supplying drawing parameters
*                pixmap - X pixmap to draw on
*   Returns:     alters pixmap.
*/

```

```
void  UpdatePoint(dpy,frame,pixmap)
```

```
Display      *dpy;
```

```
Frame frame;
```

```
Pixmap      pixmap;
```

```

{
    unsigned long      gcmask;
    XGCValues  gcvals;
    GC  gc;
    Video vid=frame->video;
    int  posn[2]={frame->point->location[0],frame->point->location[1]},
channel=3==frame->channel?0:frame->channel;

    gcvals.function=GXequiv;
    gcmask=GCFFunction;

```

- 366 -

```

gcvals.foreground = 127;
gcmask = gcmask | GCForeground;
gc = XCreateGC(dpy, pixmap, gcmask, &gcvals);
if (vid->type == YUV && channel != 0) {
    posn[0] = posn[0] >> vid->UVsample[0];
    posn[1] = posn[1] >> vid->UVsample[1];
}
if (vid->trans.type != TRANS_Wave) {

CrossHair(dpy, pixmap, gc, 0, 0, Size(vid, channel, 0), Size(vid, channel, 1), posn[0], posn[1], fra
me->zoom);
    } else {
        int    octs = vid->trans.wavelet.space[vid->type == YUV &&
channel != 0 ? 1 : 0], oct,
                size[2] = {Size(vid, channel, 0), Size(vid, channel, 1)};

CrossHair(dpy, pixmap, gc, 0, 0, size[0], size[1], posn[0], posn[1], frame->zoom-octs);
        for(oct = 1; oct <= octs; oct++) {

CrossHair(dpy, pixmap, gc, size[0], 0, size[0], size[1], posn[0], posn[1], frame->zoom-oct);

CrossHair(dpy, pixmap, gc, 0, size[1], size[0], size[1], posn[0], posn[1], frame->zoom-oct);

CrossHair(dpy, pixmap, gc, size[0], size[1], size[0], size[1], posn[0], posn[1], frame->zoom-oct
);
        }
    }
XFreeGC(dpy, gc);
}

```

- 367 -

source/Video2.c

/*

Video callback routines for Listing, Loading

*/

#include "../include/xwave.h"

#include "../include/ImageHeader.h"

#include "../include/DTheader.h"

#include "Video.h"

#include <sys/time.h>

extern void EraseFrame();

extern void CvtIndex();

void SortList(list,no)

String list[];

int no;

{

int i, j, k;

if (no > 1) for(i=1; i < no; i++) for(j=0; j < i; j++) {

k=0;

while(list[i][k] == list[j][k] && list[i][k] != '\0' && list[j][k] != '\0') k++;

if (list[i][k] < list[j][k]) {

String spare = list[i];

list[i] = list[j];

list[j] = spare;

}

- 368 -

```

    }
}

String *ReadDirectory(dir_path,extension)

```

```
String dir_path, extension;
```

```

{
    DIR *dirp, *opendir();
    struct dirent *dp, *readdir();
    static String *fileList=NULL, file;
    int count=0, i;
    char path[STRLEN];

    Dprintf("ReadDirectory for %s extension\n",extension);
    if (fileList!=NULL) {
        for(i=0;NULL!=fileList[i];i++) free(fileList[i]);
        free(fileList);
    }
    fileList=(String *)MALLOC(sizeof(String *)*300);
    sprintf(path, "%s%s\0",global->home,dir_path);
    dirp = opendir(path);
    for (dp=readdir(dirp);dp!=NULL && count<299;dp=readdir(dirp)) {
        int length=strlen(dp->d_name);

        if (length>=strlen(extension))
            if (!strcmp(dp->d_name+length-strlen(extension),extension)) {
                Dprintf("Found %s in dir\n",dp->d_name);
                fileList[count]=(char *)MALLOC(length+1);
                strncpy(fileList[count],dp->d_name,length-strlen(extension));
                count+=1;
            }
    }
}

```

- 369 -

```

    }
    fileList[count]=NULL;
    SortList(fileList,count);
    closedir(dirp);
    return(fileList);
}

int  Shift(value,shift)

int  value, shift;

{
    if (shift==0) return value;
    else if (shift<0) return(value >> -shift);
    else return(value << shift);
}

int  Size(video,channel,dimension)

Video video;
int  channel, dimension;

{
    if (video->type==YUV && dimension!=2 && channel!=0 && channel!=3)
return(video->size[dimension]>> video->UVsample[dimension]);
    else return(video->size[dimension]);
}

int  Address2(video,channel,x,y)

Video video;
int  channel, x, y;

```

- 370 -

```
{
    if (video->type == YUV && channel!=0 && channel!=3)
return(x+Size(video,channel,0)*y);
    else return(x+video->size[0]*y);
}
```

```
int    Address(video,channel,x,y)
```

```
Video video;
```

```
int    channel, x, y;
```

```
{
    if (video->type == YUV && channel!=0 && channel!=3)
return((x >> video->UVsample[0])+Size(video,channel,0)*(y >> video->UVsample[1])
);
    else return(x+video->size[0]*y);
}
```

```
String *VideoList()
```

```
{
    Dprintf("VideoList\n");
    return(ReadDirectory(VID_DIR,VID_EXT));
}
```

```
String *KlicsList()
```

```
{
    Dprintf("KlicsList\n");
    return(ReadDirectory(KLICS_DIR,KLICS_EXT));
}
```

- 371 -

```
String *KlicsListSA()
```

```
{
    Dprintf("KlicsListSA\n");
    return(ReadDirectory(KLICS_SA_DIR,KLICS_SA_EXT));
}
```

```
String *VideoCurrentList()
```

```
{
    static String videoList[300];
    Video video=global->videos;
    int count=0;

    Dprintf("VideoCurrentList\n");
    while (video!=NULL) {
        if (count==300) Dprintf("VideoCurrentList: static size exceeded\n");
        videoList[count]=video->name;
        video=video->next;
        count+=1;
    }
    videoList[count]=NULL;
    SortList(videoList,count);
    return(videoList);
}
```

```
String *VideoYUVList()
```

```
{
    static String videoList[300];
    Video video=global->videos;
    int count=0;
```

- 372 -

```
Dprintf("VideoCurrentList\n");
while (video!=NULL) {
    if (count==300) Dprintf("VideoYUVList: static size exceeded\n");
    if (video->type==YUV) videoList[count++] = video->name;
    video=video->next;
}
videoList[count]=NULL;
SortList(videoList,count);
return(videoList);
}
```

```
String *VideoDropList()
```

```
{
    static String videoList[300];
    Video video=global->videos;
    int count=0;
    Boolean VideoHasFrame();

    Dprintf("VideoDropList\n");
    while (video!=NULL) {
        if (False==VideoHasFrame(video,global->frames)) {
            videoList[count]=video->name;
            count+=1;
        };
        video=video->next;
    }
    videoList[count]=NULL;
    SortList(videoList,count);
    return(videoList);
}
```

- 373 -

Boolean VideoHasFrame(video.frame)

Video video;

Frame frame;

```
{
    if (frame == NULL) return(False);
    else if (frame->video == video) return(True);
    else return(VideoHasFrame(video, frame->next));
}
```

void VideoLoad(w, closure, call_data)

Widget w;

caddr_t closure, call_data;

```
{
    Video vid=(Video)MALLOC(sizeof(VideoRec));
    XawListReturnStruct *name=(XawListReturnStruct *)call_data;
    int    frame, channel;

    Dprintf("VideoLoad %s\n", name->string);
    strcpy(vid->name, name->string);
    strcpy(vid->files, name->string);
    vid->next=global->videos;
    global->videos=vid;
    vid->rate=30;
    Parse(VID_DIR, name->string, VID_EXT);
    for (channel=0; channel < (vid->type == MONO?1:3); channel++)
        vid->data[channel]=(short **)MALLOC(sizeof(short *)*vid->size[2]);
    if (!vid->disk) for(frame=0; frame < vid->size[2]; frame++)
        GetFrame(vid, frame);
}
```

- 374 -

```

Dprintf("VideoLoad terminated\n");
if (global->batch == NULL) InitFrame(w.closure, call_data);
}

void VideoSave(w, closure, call_data)

Widget      w;
caddr_t     closure, call_data;

{
    Video video;
    XawListReturnStruct *name = (XawListReturnStruct *)call_data;
    int frame;

    video = FindVideo(name->string, global->videos);
    if (video->files[0] == '\0') strcpy(video->files, name->string);
    SaveHeader(video);
    for (frame = 0; frame < video->size[2]; frame++) {
        Boolean disk = video->disk;

        GetFrame(video, frame);
        video->disk = True;
        SaveFrame(video, frame);
        video->disk = disk;
        FreeFrame(video, frame);
    }
    Dprintf("VideoSave terminated\n");
}

void VideoDTSave(w, closure, call_data)

Widget      w;

```

- 375 -

```
caddr_t      closure, call_data;
```

```
{
    Video video;
    FILE *fp, *fopen();
    XawListReturnStruct *name = (XawListReturnStruct *)call_data;
    char  file_name[STRLEN], whole_frame[512][512];
    int   frame, i, x, y, offset[2];
    DTheader
header = {"DT-IMAGE", 1, 4, 1, 2, "", "", 1, {0, 0, 4, 0}, 1, 1, 0, 1, {4, 3}, 8, 1, {0, 2}, {0, 2}, {0, 2}, {0, 2}, "", "xwave generated image", ""};

    Dprintf("VideoDTSave %s\n", name->string);
    video = FindVideo(name->string, global->videos);

    sprintf(file_name, "%s%s/%s/%s%s\0", global->home, IMAGE_DIR, video->path, video->files, ".img");

    offset[0] = (512 - video->size[0]) / 2;
    offset[1] = (512 - video->size[1]) / 2;
    offset[0] = offset[0] < 0 ? 0 : offset[0];
    offset[1] = offset[1] < 0 ? 0 : offset[1];
    fp = fopen(file_name, "w");
    fwrite(&header, 1, sizeof(DTheader), fp);
    GetFrame(video, 0);
    for(y = 0; y < 512; y++) for(x = 0; x < 512; x++) {
        int    X, Y, oct;

        if (y < offset[1] || x < offset[0] || y - offset[1] > video->size[1] ||
x - offset[0] > video->size[0]) whole_frame[y][x] = 0;
        else {
            if (video->trans.type == TRANS_Wave) {
```


- 376 -

```
CvtIndex(x-offset[0],y-offset[1],video->size[0],video->size[1],video->trans.wavelet.space[0],&X.&Y.&oct);
```

```
whole_frame[y][x]=128+Round(video->data[0][0][Y*video->size[0]+X]*(oct==video->trans.wavelet.space[0]?1:4),video->precision);
```

```
    } else {
```

```
        X=x-offset[0]; Y=y-offset[1];
```

```
whole_frame[y][x]=128+Round(video->data[0][0][Y*video->size[0]+X],video->precision);
```

```
    }
```

```
  }
```

```
}
```

```
FreeFrame(video,0);
```

```
fwrite(whole_frame,1,512*512,fp);
```

```
fclose(fp);
```

```
}
```

```
void VideoXimSave(w,closure,call_data)
```

```
Widget      w;
```

```
caddr_t     closure, call_data;
```

```
{
```

```
    Video video;
```

```
    FILE *fp, *fopen();
```

```
    XawListReturnStruct *name=(XawListReturnStruct *)call_data;
```

```
    char file_name[STRLEN], *whole_frame;
```

```
    int frame, channel, i, x, y;
```

```
    ImageHeader header;
```

```
Dprintf("VideoXimSave %s\n",name->string);
```

- 377 -

```

video = FindVideo(name->string, global->videos);
whole_frame = (char *)MALLOC(video->size[0]*video->size[1]);
if (video->files[0] == '\0') strcpy(video->files.name->string);

sprintf(file_name, "%s%s/%s/%s%s\0", global->home, IMAGE_DIR, video->path, video-
>files, ".xim");
fp = fopen(file_name, "w");
sprintf(header.file_version, "%8d", IMAGE_VERSION);
sprintf(header.header_size, "%8d", 1024);
sprintf(header.image_width, "%8d", video->size[0]);
sprintf(header.image_height, "%8d", video->size[1]);
sprintf(header.num_colors, "%8d", 256);
sprintf(header.num_channels, "%8d", video->type == MONO?1:3);
sprintf(header.num_pictures, "%8d", video->size[2]);
sprintf(header.alpha_channel, "%4d", 0);
sprintf(header.runlength, "%4d", 0);
sprintf(header.author, "%48s", "xwave");
sprintf(header.date, "%32s", "Now");
sprintf(header.program, "%16s", "xwave");
for(i=0; i<256; i++) {
    header.c_map[i][0] = (unsigned char)i;
    header.c_map[i][1] = (unsigned char)i;
    header.c_map[i][2] = (unsigned char)i;
}
fwrite(&header, 1, sizeof(ImageHeader), fp);
for (frame = video->start; frame < video->start + video->size[2]; frame++) {
    GetFrame(video, frame-video->start);
    for(channel=0; channel < (video->type == MONO?1:3); channel++) {
        for(x=0; x < video->size[0]; x++)
            for(y=0; y < video->size[1]; y++)

whole_frame[x + video->size[0]*y] = itc(video->data[channel][frame-video->start][Addre

```

- 378 -

```

ss(video.channel.x,y)] > > video->precision);
        fwrite(whole_frame,sizeof(char),video->size[0]*video->size[1],fp);
    }
    FreeFrame(video.frame-video->start);
}
fclose(fp);
XtFree(whole_frame);
}

```

```

void VideoMacSave(w,closure,call_data)

```

```

Widget      w;

```

```

caddr_t      closure, call_data;

```

```

{

```

```

    Video video;

```

```

    FILE *fp, *fopen();

```

```

    XawListReturnStruct *name=(XawListReturnStruct *)call_data;

```

```

    char file_name[STRLEN], *whole_frame;

```

```

    int frame, channel, i, x, y;

```

```

    Dprintf("VideoMacSave %s\n",name->string);

```

```

    video=FindVideo(name->string,global->videos);

```

```

    if (video->files[0] == '\0') strcpy(video->files,name->string);

```

```

    sprintf(file_name,"%s%s/%s/%s%s\0",global->home.IMAGE_DIR,video->path,video-
    >files, ".mac");

```

```

    fp=fopen(file_name,"w");

```

```

    whole_frame=(char *)MALLOC(video->size[1]*video->size[0]*3);

```

```

    for(frame=0;frame<video->size[2];frame++) {

```

```

        int size=video->size[0]*video->size[1];

```

- 379 -

```

    GetFrame(video,frame);
    for(channel=0;channel < (video->type == MONO?1:3);channel++)
        for(x=0;x < video->size[0];x++)
            for(y=0;y < video->size[1];y++)

whole_frame[(x+video->size[0]*y)*3+channel]=itc(video->data[channel][frame][Address(video.channel,x,y)] >> video->precision);
    fwrite(whole_frame,1,3*size,fp);
    FreeFrame(video,frame);
}
fclose(fp);
XtFree(whole_frame);
}

```

```
void VideoHexSave(w,closure,call_data)
```

```
Widget      w;
```

```
caddr_t     closure, call_data;
```

```
{
```

```
    Video video;
```

```
    FILE *fp, *fopen();
```

```
    XawListReturnStruct *name=(XawListReturnStruct *)call_data;
```

```
    char file_name[STRLEN];
```

```
    int frame, channel, i;
```

```
    Dprintf("VideoHexSave %s\n",name->string);
```

```
    video=FindVideo(name->string,global->videos);
```

```
    if (video->files[0] == '\0') strcpy(video->files,name->string);
```

```
    sprintf(file_name,"%s%s/%s/%s%s\0",global->home,IMAGE_DIR,video->path,video->files,".h");
```

- 380 -

```

fp = fopen(file_name, "w");
for(frame=0; frame < (video->size[2] > 2?2:video->size[2]); frame++) {
    int    size = video->size[1]*video->size[0];

    GetFrame(video, frame);
    fprintf(fp, "char
%s %d[ %d] = {\n", name->string[strlen(name->string)-1] == 'd'? "src": "dst", frame.size);
    for(i=0; i < size; i++)

fprintf(fp, "0x %02x, %c", (video->data[0][frame][i] > > video->precision) + 128, i%20 ==
19? '\n': ' ');

    fprintf(fp, "\n};\n");
    FreeFrame(video, frame);
}
fclose(fp);
}

```

```
#define AB_WIDTH 1440
```

```
#define AB_HEIGHT 486
```

```
void VideoAbekusSave(w, closure, call_data)
```

```
Widget      w;
```

```
caddr_t     closure, call_data;
```

```

{
    AbekusCtrl ctrl = (AbekusCtrl)closure;
    FILE *fp, *fopen();
    char file_name[STRLEN], *data = (char
*)MALLOC(AB_WIDTH*AB_HEIGHT), zero = itc(0);
    int frame, channel, i, x, y, length=0;
    Video vids[4];

```

- 381 -

```

Dprintf("VideoAbekusSave\n");
for(i=0;i<4;i++)
    if (ctrl->names[i]!=NULL) {
        vids[i]=FindVideo(ctrl->names[i],global->videos);
        length=length>vids[i]->size[2]?length:vids[i]->size[2];
    } else vids[i]=NULL;
for(frame=0;frame<length;frame++) {
    sprintf(file_name,"%d.yuv\0",frame+1);
    fp=fopen(file_name,"w");
    for(i=0;i<4;i++) GetFrame(vids[i],frame);
    for(y=0;y<AB_HEIGHT;y++)
        for(x=0;x<AB_WIDTH;x++) {
            int
i=(x<AB_WIDTH/2?0:1)+(y<AB_HEIGHT/2?0:2),
            Y=y<AB_HEIGHT/2?y:y-AB_HEIGHT/2,
            X=(x<AB_WIDTH/2?x:x-AB_WIDTH/2)/2,
            channel=((x&1)==1)?0:((X&1)==0)?1:2;

            if (vids[i]->type==MONO && channel!=0 ||
X>=vids[i]->size[0] || Y>=vids[i]->size[1]) data[x+y*AB_WIDTH]=zero;
            else
data[x+y*AB_WIDTH]=itc(vids[i]->data[channel][frame][Address(vids[i],channel,X,Y)]
>>vids[i]->precision);
        }
    for(i=0;i<4;i++) {
        FreeFrame(vids[i],frame);
        EraseFrame(vids[i],frame);
    }
    fwrite(data,1,AB_WIDTH*AB_HEIGHT,fp);
    fclose(fp);
}
}

```

- 382 -

```
void VideoDrop(w,closure,call_data)
```

```
Widget      w;
```

```
caddr_t     closure, call_data;
```

```
{
```

```
Video *videos=&global-> videos, video;
```

```
XawListReturnStruct *name=(XawListReturnStruct *)call_data;
```

```
int  channel, frame;
```

```
Dprintf("VideoDrop %s\n",name-> string);
```

```
video=FindVideo(name-> string,global-> videos);
```

```
while (*videos!=video && *videos!=NULL) videos=&((*videos)-> next);
```

```
if (*videos!=NULL) {
```

```
    *videos=(*videos)-> next;
```

```
    for(channel=0;channel<(video-> type == MONO?1:3);channel++)
```

```
        if (video-> data[channel]!=NULL) {
```

```
            for(frame=0;frame< video-> size[2];frame++)
```

```
                if (video-> data[channel][frame]!=NULL)
```

```
                    XtFree(video-> data[channel][frame]);
```

```
                    XtFree(video-> data[channel]);
```

```
        }
```

```
    XtFree(video);
```

```
}
```

```
}
```

```
/* Obsolete
```

```
void VideoDiff(w,closure,call_data)
```

```
Widget      w;
```

```
caddr_t     closure, call_data;
```

```
{
```

- 383 -

```

XawListReturnStruct *name = (XawListReturnStruct *)call_data;
Video src = FindVideo(name->string, global->videos), dst = CopyHeader(src);
int frame, channel, i;

printf("VideoDiff %s\n", name->string);
sprintf(dst->name, "%s.diff\0", src->name);
for(frame = 0; frame < src->size[2]; frame++) {
    GetFrame(src, frame);
    NewFrame(dst, frame);
    for(channel = 0; channel < (video->type == MONO?1:3); channel++)
        for(i = 0; i < src->size[1]*src->size[0]; i++)

dst->data[channel][frame][i] = src->data[channel][frame][i] - (frame == 0?0:src->data[channel][frame-1][i]);
    SaveFrame(dst, frame);
    FreeFrame(dst, frame);
    if (frame > 0) FreeFrame(src, frame-1);
}
FreeFrame(dst, src->size[2]-1);
dst->next = global->videos;
global->videos = dst;
}
*/
void VideoClean(w, closure, call_data)

Widget w;
caddr_t closure, call_data;

{
    Video *videos = &global->videos, video;
    int channel, frame;

```



```

Dprintf("VideoClean\n");
while(*videos!=NULL) {
    video=*videos;
    if (False == VideoHasFrame(video,global->frames)) {
        Dprintf("Erasing video: %s\n",video->name);

for(channel=0;channel<(video->type==MONO?1:3);channel++)
        if (video->data[channel]!=NULL) {
            for(frame=0;frame<video->size[2];frame++)
                if (video->data[channel][frame]!=NULL)
XtFree(video->data[channel][frame]);
            XtFree(video->data[channel]);
        }
        *videos=video->next;
        XtFree(video);
    } else videos=&(*videos)->next;
}
}

```

```

typedef struct {
    Frame frame;
    XtIntervalId id;
    unsigned long interval;
    long msec, shown, average;
    Pixmap *movie;
    int fno, old_fno;
} MovieArgRec, *MovieArg;

```

```

void Projector(client_data,id)

```

```

XtPointer client_data;
XtIntervalId *id;

```

- 385 -

```

{
    MovieArg    movieArg=(MovieArg)client_data;
    Display      *dpy=XtDisplay(global->toplevel);
    struct timeval    tp;
    struct timezone    tzp;
    long    new_msec;
    int    scrn=XDefaultScreen(dpy);

movieArg->id=XtAppAddTimeOut(global->app_con,movieArg->interval,Projector,mo
vieArg);

    gettimeofday(&tp,&tzp);
    new_msec=tp.tv_sec*1000+tp.tv_usec/1000;
    if (movieArg->msec!=0) {

movieArg->average=(movieArg->average*movieArg->shown+new_msec-movieArg-
>msec)/(movieArg->shown+1);
        movieArg->shown++;
    }
    movieArg->msec=new_msec;

XCopyArea(dpy,movieArg->movie[movieArg->fno],XtWindow(movieArg->frame->i
mage_widget),DefaultGC(dpy,scrn),0,0,movieArg->frame->video->size[0],movieArg-
>frame->video->size[1],0,0);

movieArg->fno=movieArg->fno==movieArg->frame->video->size[2]-1?0:movieAr
g->fno+1;
}

void    StopMovie(w,closure,call_data)

Widget    w;

```

- 386 -

```
caddr_t      closure, call_data;
```

```
{
```

```
    MovieArg    movieArg=(MovieArg)closure;
```

```
    Display      *dpy=XtDisplay(global->toplevel);
```

```
    int          i;
```

```
    Arg          args[1];
```

```
    XtRemoveTimeout(movieArg->id);
```

```
    Dprintf("Movie showed %d frames at an average of %f  
fps\n",movieArg->shown,1000.0/(float)movieArg->average);
```

```
    for(i=0;i<movieArg->frame->video->size[2];i++)
```

```
    XFreePixmap(dpy,movieArg->movie[i]);
```

```
    XtFree(movieArg->movie);
```

```
    XtSetArg(args[0],XtNbitmap,UpdateImage(movieArg->frame));
```

```
    XtSetValues(movieArg->frame->image_widget,args,ONE);
```

```
    XSynchronize(dpy,False);
```

```
}
```

```
#define      MOVIE_ICONS      1
```

```
void  Movie(w,closure,call_data)
```

```
Widget      w;
```

```
caddr_t      closure, call_data;
```

```
{
```

```
    Video video=((Frame)closure->video);
```

```
    MovieArg    movieArg=(MovieArg)MALLOC(sizeof(MovieArgRec));
```

```
    Widget      shell=ShellWidget("movie",XtParent(w),SW_over,NULL,NULL),
```

```
               form=FormatWidget("movie_form",shell),
```

```
    widgets[MOVIE_ICONS];
```

- 387 -

```

Display      *dpy=XtDisplay(global->toplevel);
FormItem     items[]={
    {"movie_stop","stop",0,0,FW_icon,NULL},
};
XtCallbackRec callbacks[]={
    {StopMovie,(caddr_t)movieArg},
    {Free,(caddr_t)movieArg},
    {Destroy,(caddr_t)shell},
    {NULL,NULL},
};
int          i;
XGCValues    values;
GC           gc;

Dprintf("Movie\n");

FillForm(form,MOVIE_ICONS,items,widgets,callbacks);
XtPopup(shell,XtGrabExclusive);

values.foreground=255;
values.background=0;
gc=XtGetGC(XtParent(w),GCForeground | GCBackground,&values);
movieArg->frame=(Frame)closure;
movieArg->movie=(Pixmap *)MALLOC(video->size[2]*sizeof(Pixmap));
movieArg->old_fno=movieArg->frame->frame;
for(i=0;i<video->size[2];i++) {
    char  fno[STRLEN];

    sprintf(fno,"%03d\0",i+video->start);
    movieArg->frame->frame=i;
    GetFrame(video,i);
    movieArg->movie[i]=UpdateImage(movieArg->frame);
}

```

- 388 -

```

XDrawImageString(dpy, movieArg->movie[i], gc, video->size[0]-50, 10, fno, 3);

XCopyArea(dpy, movieArg->movie[i], XtWindow(movieArg->frame->image_widget), D
efaultGC(dpy, 0), 0, 0, video->size[0], video->size[1], 0, 0);
    movieArg->frame->frame = movieArg->old_fno;
    FreeFrame(video, i);
}
XtDestroyGC(gc);
movieArg->fno = 0;
movieArg->msec = 0;
movieArg->shown = 0;
movieArg->average = 0;
movieArg->interval = 1000/video->rate;

movieArg->id = XtAppAddTimeOut(global->app_con, movieArg->interval, Projector, mo
vieArg);
    XSynchronize(dpy, True);
}

void Compare(w, closure, call_data)

Widget      w;
caddr_t     closure, call_data;

{
    XawListReturnStruct *name = (XawListReturnStruct *)call_data;
    Video src = (Video)closure, dst = FindVideo(name->string, global->videos);
    int channels = src->type == MONO || dst->type == MONO ? 1 : 3, channel,
    values = 0, x, y,
        frames = src->size[2] > dst->size[2] ? dst->size[2] : src->size[2],
    frame;

```

- 389 -

```

double      mse;
Message      msg = NewMessage(NULL,400);
XtCallbackRec  callbacks[] = {
    {CloseMessage, (caddr_t)msg}, {NULL, NULL},
};

msg->rows = frames > 5?10:2*frames; msg->cols = 40;
if (global->batch == NULL)
MessageWindow(FindWidget("frm_compare",w),msg,"Compare",True,callbacks);
for(frame=0;frame < frames;frame++) {
    Boolean      srcp = src->precision > dst->precision;
    int          err_sqr=0,
precision = srcp?src->precision-dst->precision:dst->precision-src->precision;

    Mprintf(msg,"Compare: %s%03d and
%s%03d\n",src->name,src->start+frame,dst->name,dst->start+frame);
    GetFrame(src,frame);
    GetFrame(dst,frame);
    for(channel=0;channel < channels;channel++) {

values += Size(src->size[1] > dst->size[1]?dst:src,channel,1)*Size(src->size[0] > dst->s
ize[0]?dst:src,channel,0);

for(y=0;y < Size(src->size[1] > dst->size[1]?dst:src,channel,1);y++)

for(x=0;x < Size(src->size[0] > dst->size[0]?dst:src,channel,0);x++) {
        int
err = (src->data[channel][frame][x+Size(src,channel,0)*y] < < (srcp?0:precision))-(dst->
data[channel][frame][x+Size(dst,channel,0)*y] < < (srcp?precision:0));

        err_sqr += err*err;
    }
}

```

- 390 -

```
    }  
    FreeFrame(src.frame);  
    FreeFrame(dst.frame);  
    mse = (double)err_sqr/(double)(values);  
    Mprintf(msg, "Error %d MSE %f PSNR  
%f\n", err_sqr.mse, 10*log10(pow((pow(2.0,(double)(8+(srcp?src->precision:dst->precis  
ion)))-1),2.0)/mse));  
    Mflush(msg);  
    }  
}
```

```
void BatchCompare(w,closure,call_data)
```

```
Widget      w;  
caddr_t     closure, call_data;  
  
{  
    String name = (String)closure;  
  
    closure = (caddr_t)FindVideo(name,global->videos);  
    Compare(w,closure,call_data);  
}
```

- 391 -

source/xwave.c

```
#include    "../include/xwave.h"
#include    <X11/Xresource.h>
#include    <X11/Intrinsic.h>
#include    <X11/Quarks.h>
```

```
extern Palette    ReOrderPalettes();
extern void    NameButton();
extern void    ImageNotify();
extern void    Parse();
```

```
#define    IconPath    "bitmaps"
#define    IconFile    "xwave.icons"
#define    CompressPath    "."
#define    CompressExt    ".compress"
#define    PalettePath    "."
#define    PaletteExt    ".pal"
```

```
Global    global;
```

```
String ChannelName[3][4]={
    {"GreyScale",NULL,NULL,NULL},
    {"Red ", "Green", "Blue ", "Color"},
    {"Y-Luminance", "U-Chrome ", "V-Chrome ", "Color "},
};
```

```
#define    XtNdebug    "debug"
#define    XtNbatch    "batch"
```


- 392 -

```
static XtResource resources[] = {
    {XtNdebug, XtCBoolean, XtRBoolean, sizeof(Boolean),
     XtOffset(Global.debug), XtRString, "false"},
    {XtNbatch, XtCFile, XtRString, sizeof(String),
     XtOffset(Global.batch), XtRString, NULL},
};
```

```
static XrmOptionDescRec options[]={
    {"-debug", "*debug", XrmoptionNoArg, "true"},
    {"-batch", "*batch", XrmoptionSepArg, NULL},
};
```

```
static Boolean CvtStringToPixel20;
```

```
#if defined(__STDC__)
externref XtConvertArgRec const colorConvertArgs[2];
#else
externref XtConvertArgRec colorConvertArgs[2];
#endif
```

```
static String fallback_resources[]={
    "*copy_video*Toggle*translations: #override \\n <Btn1Down>, <Btn1Up>:",
    set() notify0",
    "*copy_video*copy*state: true",
    NULL,
};
```

```
XtActionsRec actionTable[]={
    {"NameButton", NameButton},
};
```

```
main(argc, argv, envp)
```

- 393 -

```
int    argc;
char   *argv[], *envp[];

{
    void    InitPixmaps(), InitActions(), InitMain(), InitEnv(), InitDither(), Dispatch();
    GlobalRec    globalrec;

    global=&globalrec;
    global->videos=NULL;
    global->frames=NULL;
    global->points=NULL;
    InitEnv(envp);

    global->toplevel=XtAppInitialize(&(global->app_con),"xwave",options,XtNumber(options),&argc,argv,fallback_resources,NULL,ZERO);

    XtGetApplicationResources(global->toplevel,global,resources,XtNumber(resources),NULL,ZERO);
    if (global->batch!=NULL) {
        Parse(BATCH_DIR,global->batch,BATCH_EXT);
        if (global->batch_list!=NULL) Dispatch(global->batch_list);
    }
    if (global->batch==NULL) {
        XtAppAddActions(global->app_con,actionTable,XtNumber(actionTable));

    XtSetTypeConverter(XtRString,XtRPixel,CvtStringToPixel2,colorConvertArgs,XtNumber(colorConvertArgs),XtCacheByDisplay,NULL);
        if (global->debug) Dprintf("Xwave Debugging Output\n");
        InitVisual();
        InitDither();
        InitPixmaps(IconPath,IconFile);
        Parse(PalettePath,"xwave",PaletteExt);
    }
```

- 394 -

```
global->palettes = ReOrderPalenes(global->palenes,global->palenes);
InitActions(global->app_con);
InitMain();
XtRealizeWidget(global->toplevel);
XtAppMainLoop(global->app_con);
}
}
```

```
void InitEnv(envp)
```

```
char *envp[];
```

```
{
    String home=NULL, xwave=NULL;

    Dprintf("Initializing enviroment\n");
    while(*envp!=NULL) {
        if(!strcmp(*envp,"HOME=",5)) home=(*envp)+5;
        if(!strcmp(*envp,"XWAVE=",6)) xwave=(*envp)+6;
        envp++;
    }
    if (xwave!=NULL) sprintf(global->home,"%s/",xwave);
    else sprintf(global->home,"%s/xwave/",home);
}
```

```
#define HEIGHT 14
```

```
void InitPixmap(path,file)
```

```
char *file, *path;
```

```
{
```

- 395 -

```

FILE *fp, *fopen();
Icon icons;
char pad[100];
Display *dpy = XtDisplay(global-> toplevel);
int i, j, sink, scrn = XDefaultScreen(dpy), depth = DisplayPlanes(dpy, scrn),
    bpl = (global-> levels * depth + 7) / 8;
char data[HEIGHT * bpl];
XImage
*image = XCreateImage(dpy, global-> visinfo-> visual, depth, ZPixmap, 0, data, global-> levels, HEIGHT, 8, bpl);

sprintf(pad, "%s%s/%s\0", global-> home, path, file);
if (NULL == (fp = fopen(pad, "r"))) {
    Eprintf("Can't open file %s\n", pad);
    exit();
}
fscanf(fp, "%d\n", &global-> no_icons);
global-> icons = (Icon) MALLOC((1 + global-> no_icons) * sizeof(IconRec));
for(i = 0; i < global-> no_icons; i++) {
    global-> icons[i].name = (String) MALLOC(100);
    fscanf(fp, "%s\n", global-> icons[i].name);
    sprintf(pad, "%s%s/%s\0", global-> home, path, global-> icons[i].name);
    XReadBitmapFile(
        XtDisplay(global-> toplevel),
        XDefaultRootWindow(dpy),
        pad,
        &global-> icons[i].width,
        &global-> icons[i].height,
        &global-> icons[i].pixmap,
        &sink,
        &sink
    );
}

```

- 396 -

```

    }
    global->icons[global->no_icons].name=(String)MALLOC(100);
    strcpy(global->icons[global->no_icons].name,"colors");
    global->icons[global->no_icons].width=global->levels;
    global->icons[global->no_icons].height=HEIGHT;
    for(i=0;i < global->levels;i++)
        for(j=0;j < HEIGHT;j++) XPutPixel(image,i,j,i);

    global->icons[global->no_icons].pixmap=XCreatePixmap(dpy,XDefaultRootWindow(dp
y),global->levels,HEIGHT,depth);

    XPutImage(dpy,global->icons[global->no_icons].pixmap,DefaultGC(dpy,scrn),image,0,0
,0,0,global->levels,HEIGHT);
    global->no_icons++;
    XFree(image);
    fclose(fp);
}

#define done(type, value) \
{ \
    if (toVal->addr != NULL) { \
        if (toVal->size < sizeof(type)) { \
            toVal->size = sizeof(type); \
            return False; \
        } \
        *(type*)(toVal->addr) = (value); \
    } \
    else { \
        static type static_val; \
        static_val = (value); \
        toVal->addr = (XtPointer)&static_val; \
    } \
}

```

- 397 -

```

        toVal->size = sizeof(type);
        return True;
    }

#define      dist(colora,colorb) \

abs(colora.red-colorb.red)+abs(colora.green-colorb.green)+abs(colora.blue-colorb.blue)

static Boolean CvtStringToPixel2(dpy, args, num_args, fromVal, toVal, closure_ret)
    Display* dpy;
    XrmValuePtr args;
    Cardinal *num_args;
    XrmValuePtr fromVal;
    XrmValuePtr toVal;
    XtPointer *closure_ret;
{
    String      str = (String)fromVal->addr;
    XColor      screenColor;
    XColor      exactColor;
    Screen      *screen;
    Colormap    colormap;
    Status      status;
    String      params[1];
    Cardinal     num_params=1;

    Dprintf("Convert string to pixel 2\n");
    if (*num_args != 2)
        XtAppErrorMsg(XtDisplayToApplicationContext(dpy), "wrongParameters",
            "cvtStringToPixel",
                "XtToolkitError",
                "String to pixel conversion needs screen and colormap arguments",
                (String *)NULL, (Cardinal *)NULL);

```

- 398 -

```

screen = *((Screen **) args[0].addr);
colormap = *((Colormap *) args[1].addr);

if (!strcmp(str,XtDefaultBackground)) {
    *closure_ret = False;
    done(Pixel,WhitePixelOfScreen(screen));
}
if (!strcmp(str,XtDefaultForeground)) {
    *closure_ret = False;
    done(Pixel,BlackPixelOfScreen(screen));
}
params[0]=str;
if (0==XParseColor(DisplayOfScreen(screen),colormap,str,&screenColor)) {
    XtAppWarningMsg(XtDisplayToApplicationContext(dpy), "noColormap",
"cvStringToPixel",
    "XtToolkitError", "Cannot parse color: \"%s\"",
params,&num_params);
    return False;
} else {
    if (0==XAllocColor(DisplayOfScreen(screen),colormap,&screenColor)) {
        int i, delta, closest=0;
        XColor colors[global->levels];

        for(i=0;i<global->levels;i++) colors[i].pixel=i;

XQueryColors(DisplayOfScreen(screen),colormap,colors,global->levels);
        delta=dist(screenColor,colors[0]);
        for(i=1;i<global->levels;i++) {
            int delta_new=dist(screenColor,colors[i]);

            if (delta_new<delta) {
                delta=delta_new;

```

- 399 -

```

        closest = i;
    }
}
Dprintf("Closest color to %s is pixel %d red %d green %d blue
%d\n", str, colors[closest].pixel, colors[closest].red, colors[closest].green, colors[closest].blue
);
    *closure_ret = (char*)True;
    done(Pixel, closest);
} else {
    *closure_ret = (char*)True;
    done(Pixel, screenColor.pixel);
}
}
}

```

```
void Dispatch(list)
```

```
Batch list;
```

```

{
    if (list->next != NULL) Dispatch(list->next);
    (list->proc)(NULL, list->closure, list->call_data);
    if (list->closure != NULL) XtFree(list->closure);
    if (list->call_data != NULL) XtFree(list->call_data);
    XtFree(list);
}

```

```
void BatchCtrl(w, closure, call_data)
```

```
Widget      w;
```

```
caddr_t     closure, call_data;
```


- 400 -

```
{
    Dprintf("BatchCtrl\n");
    global->batch=(String)closure;
}

void UnixShell(w,closure,call_data)

Widget      w;
caddr_t     closure, call_data;

{
    if (-1 == Fork((char **)closure)) Eprintf("Unable to fork\n");
}

void InitDither()

{
    int      i, j, k, l,
             dm4[4][4]={
                0, 8, 2, 10,
                12, 4, 14, 6,
                3, 11, 1, 9,
                15, 7, 13, 5
            };

    for(i=0;i<4;i++)
        for(j=0;j<4;j++)
            for(k=0;k<4;k++)
                for(l=0;l<4;l++)

global->dither[4*k+i][4*l+j]=(dm4[i][j]<<4)+dm4[k][l];
}
```

- 401 -

source/Copy.h

```
typedef struct {  
    Video video;  
    char name[STRLEN], src_name[STRLEN];  
    int UVsample[2];  
    int mode;  
    Widget radioGroup;  
} CopyCurlRec, *CopyCtrl;
```

- 402 -

source/Gram.y

%{

/*

* Grammar for files: .elo

*/

#include "../include/xwave.h"

#include "Klics.h"

#include "Transform.h"

#include "Copy.h"

#include "Video.h"

extern void VideoLoad();

extern void VideoSave();

extern void VideoDrop();

extern void ImportKlics();

extern void VideoAbekusSave();

extern void UnixShell();

extern void BatchCompCtrl();

extern void BatchTransCtrl();

extern void BatchCopyCtrl();

extern void BatchCompare();

extern void BatchCtrl();

extern CompCtrl InitCompCtrl();

extern CopyCtrl InitCopyCtrl();

extern TransCtrl InitTransCtrl();

static char *ptr;

void NewBatch();

%}

- 403 -

%union

{

double fnum;

int num;

char *ptr;

Boolean bool;

};

%token SIZE TRANSFORM TRANSFORM_NONE TRANSFORM_WAVE PATH

%token FILE_PAL PALETTE RANGE LINE

%token FILE_VID TYPE FORMAT_MONO FORMAT_RGB FORMAT_YUV

RATE DISK GAMMA PATH FILES START END LEN DIM HEADER OFFSETS

NEGATIVE PRECISION

%token FILE_BAT LOAD SAVE SAVE_ABEKUS COMPARE DROP

COMPRESS VIDEO_NAME STATS_NAME BIN_NAME

%token STILL_MODE VIDEO_MODE AUTO_Q QUANT_CONST

THRESH_CONST BASE_FACTOR DIAG_FACTOR CHROME_FACTOR

%token DECISION DEC_MAX DEC_SIGABS DEC_SIGSQR FEEDBACK

FILTER FLT_NONE FLT_EXP CMP_CONST SPACE LEFT_BRACE RIGHT_BRACE

DIRECTION

%token FPS BITRATE BUFFER XWAVE SHELL IMPORT_KLICs

%token COPY DIRECT_COPY DIFF LPF_WIPE LPF_ONLY RGB_YUV

%token < num > NUMBER

%token < ptr > STRING

%token < fnum > FNUMBER

%token < bool > BOOLEAN

%type < num > number video_type decision filter

%type < ptr > string

%type < fnum > fnumber

%type < bool > boolean

- 404 -

%start wait

% %

wait :

```

| pal_id pal_desc
| video_id video_desc
| bat_id bat_desc bat_end;

```

pal_id : FILE_PAL {

Dprintf("Gram: palette file %s\n",global->parse_file);

};

video_id : FILE_VID {

Dprintf("Gram: video file %s\n",global->parse_file);

global->videos->start=1;

global->videos->size[2]=1;

};

bat_id : FILE_BAT {

Dprintf("Gram: batch file %s\n",global->parse_file);

};

pal_desc :

| pal_desc palette LEFT_BRACE mappings RIGHT_BRACE;

palette : PALETTE string {

Palette pal=(Palette)MALLOC(sizeof(PaletteRec));

Dprintf("Gram: palette %s\n", \$2);

strcpy(pal->name,\$2);

pal->mappings=NULL;

- 405 -

```
pal->next=global->palettes;
```

```
global->palettes=pal;
```

```
global->no_pals++;
```

```
};
```

```
mappings
```

```
:
```

```
| mappings mapping;
```

```
mapping
```

```
: RANGE number number LINE number number {
```

```
Map map=(Map)MALLOC(sizeof(MapRec));
```

```
Dprintf("Gram: Range %d to %d m=%d c=%d\n", $2, $3, $5, $6);
```

```
map->start=$2;
```

```
map->finish=$3;
```

```
map->m=$5;
```

```
map->c=$6;
```

```
map->next=global->palettes->mappings;
```

```
global->palettes->mappings=map;
```

```
};
```

```
video_desc : video_defs {
```

```
if (global->videos->size[0]==0 &&
```

```
global->videos->size[1]==0) {
```

```
global->videos->size[0]=global->videos->cols;
```

```
global->videos->size[1]=global->videos->rows;
```

```
}
```

```
};
```

```
video_defs :
```

```
| video_defs video_def;
```

```
video_def : PATH string {
```

- 406 -

```
Dprintf("Video path %s\n", $2);
strcpy(global->videos->path, $2);
}
| FILES string {
    Dprintf("Frames stored in %s\n", $2);
    strcpy(global->videos->files, $2);
}
| TYPE video_type {
    String types[] = {"Mono", "RGB", "YUV"};

    Dprintf("Video type: %s\n", types[$2]);
    global->videos->type = (VideoFormat)$2;
}
| RATE number {
    Dprintf("Video rate %d fps\n", $2);
    global->videos->rate = $2;
}
| DISK {
    Dprintf("Frames on disk\n");
    global->videos->disk = True;
}
| GAMMA {
    Dprintf("Gamma corrected\n");
    global->videos->gamma = True;
}
| NEGATIVE {
    Dprintf("Negative video\n");
    global->videos->negative = True;
}
| TRANSFORM video_transform
| START number {
    Dprintf("Video start %03d\n", $2);
```

- 407 -

```
global->videos->start=$2;
}
| END number {
    Dprintf("Video end %03d\n",$2);
    global->videos->size[2]=$2-global->videos->start+1;
}
| LEN number {
    Dprintf("Video frames %d\n",$2);
    global->videos->size[2]=$2;
}
| DIM number number {
    Dprintf("Video dimensions %d %d\n",$2,$3);
    global->videos->cols=$2;
    global->videos->rows=$3;
}
| HEADER number {
    Dprintf("Video header size %d\n",$2);
    global->videos->offset=$2;
}
| OFFSETS number number {
    Dprintf("Video offsets %d %d\n",$2,$3);
    global->videos->x_offset=$2;
    global->videos->y_offset=$3;
}
| SIZE number number {
    Dprintf("Video size %d %d\n",$2,$3);
    global->videos->size[0]=$2;
    global->videos->size[1]=$3;
}
| PRECISION number {
    Dprintf("Video precision %d bits\n",8+$2);
    global->videos->precision=$2;
```


- 408 -

```

};

video_type : FORMAT_MONO { $$=(int)MONO; }
            | FORMAT_RGB { $$=(int)RGB; }
            | FORMAT_YUV number number { $$=(int)YUV;
global->videos->UVsample[0]=$2; global->videos->UVsample[1]=$3; };

video_transform : TRANSFORM_NONE {
                global->videos->trans.type=TRANS_None;
            }
            | TRANSFORM_WAVE number number boolean {
                Dprintf("Video wavelet tranformed %d %d
%s\n", $2, $3, $4? "True": "False");
                global->videos->trans.type=TRANS_Wave;
                global->videos->trans.wavelet.space[0]=$2;
                global->videos->trans.wavelet.space[1]=$3;
                global->videos->trans.wavelet.dim=$4;
            };

bat_end :
        | XWAVE {
            Dprintf("Gram: XWAVE\n");
            NewBatch(BatchCtrl, (caddr_t) NULL, NULL);
        };

bat_desc : bat_cmds {
            Dprintf("Gram: End of batch file\n");
        };

bat_cmds :
        | bat_cmds bat_cmd;

```

- 409 -

```

bat_cmd      : simple_cmd
               | complex_cmd
               ;

simple_cmd    : LOAD string {
               XawListReturnStruct *list_return=(XawListReturnStruct
               *)MALLOC(sizeof(XawListReturnStruct));

               Dprintf("Gram: LOAD %s\n", $2);
               list_return->string = $2;
               NewBatch(VideoLoad, NULL, (caddr_t)list_return);
               }
               | SAVE string {
               XawListReturnStruct *list_return=(XawListReturnStruct
               *)MALLOC(sizeof(XawListReturnStruct));

               Dprintf("Gram: SAVE %s\n", $2);
               list_return->string = $2;
               NewBatch(VideoSave, NULL, (caddr_t)list_return);
               }
               | SAVE_ABEKUS string string string string {
               AbekusCtrl
               ctrl=(AbekusCtrl)MALLOC(sizeof(AbekusCtrlRec));

               Dprintf("Gram: SAVE_ABEKUS %s %s %s
               %s\n", $2, $3, $4, $5);

               strcpy(ctrl->names[0], $2);
               strcpy(ctrl->names[1], $3);
               strcpy(ctrl->names[2], $4);
               strcpy(ctrl->names[3], $5);
               NewBatch(VideoAbekusSave, (caddr_t)ctrl, NULL);
               }

```

- 410 -

```

| COMPARE string string {
    XawListReturnStruct *list_return=(XawListReturnStruct
*)MALLOC(sizeof(XawListReturnStruct));

    Dprintf("Gram: COMPARE %s with %s\n", $2, $3);
    list_return->string = $2;
    NewBatch(BatchCompare, (caddr_t)$3, (caddr_t)list_return);
}

| DROP string {
    XawListReturnStruct *list_return=(XawListReturnStruct
*)MALLOC(sizeof(XawListReturnStruct));

    Dprintf("Gram: DROP %s\n", $2);
    list_return->string = $2;
    NewBatch(VideoDrop, NULL, (caddr_t)list_return);
}

| IMPORT_KLICS string {
    XawListReturnStruct *list_return=(XawListReturnStruct
*)MALLOC(sizeof(XawListReturnStruct));

    Dprintf("Gram: IMPORT_KLICS %s\n", $2);
    list_return->string = $2;
    NewBatch(ImportKlics, NULL, (caddr_t)list_return);
}

| SHELL string {
    char **argv, *str = $2;
    int c, argc = 1, len = strlen(str);

    Dprintf("Shell %s\n", str);
    for(c=0; c < len; c++) if (str[c] == ' ') {
        str[c] = '\0';
        argc++;
    }
}

```

- 411 -

```

    }
    argv=(char **)MALLOC((argc+1)*sizeof(char *));
    argc=0;
    for(c=0;c < len;c += 1+strlen(str+c)) {
        argv[argc]=(char
*)MALLOC((strlen(str+c)+1)*sizeof(char));
        strcpy(argv[argc],str+c);
        argc++;
    }
    argv[argc]=NULL;
    NewBatch(UnixShell,(caddr_t)argv,NULL);
};

```

```

complex_cmd      : compress LEFT_BRACE comp_args RIGHT_BRACE
                  | transform LEFT_BRACE trans_args RIGHT_BRACE
                  | copy copy_arg;

```

```

compress        : COMPRESS string {
                  CompCtrl    ctrl=InitCompCtrl($2);

                  Dprintf("Gram: COMPRESS\n");
                  NewBatch(BatchCompCtrl,(caddr_t)ctrl,NULL);
                };

```

```

transform       : TRANSFORM string {
                  TransCtrl    ctrl=InitTransCtrl($2);

                  Dprintf("Gram: TRANSFORM\n");
                  NewBatch(BatchTransCtrl,(caddr_t)ctrl,NULL);
                };

```

```

copy           : COPY string string {

```

- 412 -

```

CopyCtrl    ctrl = InitCopyCtrl($2);
Dprintf("Gram: Copy\n");
strcpy(ctrl->name,$3);
NewBatch(BatchCopyCtrl,(caddr_t)ctrl,NULL);
};

comp_args   :
| comp_args comp_arg;

trans_args  :
| trans_args trans_arg;

copy_arg    : DIRECT_COPY number number {
    Dprintf("Gram: Direct Copy (sample %d %d)\n", $2, $3);
    ((CopyCtrl)global->batch_list->closure)->mode = 1;

    ((CopyCtrl)global->batch_list->closure)->UVsample[0] = $2;

    ((CopyCtrl)global->batch_list->closure)->UVsample[1] = $3;
}
| DIFF {
    Dprintf("Gram: Differance Copy\n");
    ((CopyCtrl)global->batch_list->closure)->mode = 2;
}
| LPF_WIPE {
    Dprintf("Gram: LPF zero\n");
    ((CopyCtrl)global->batch_list->closure)->mode = 3;
}
| LPF_ONLY {
    Dprintf("Gram: LPF only\n");
    ((CopyCtrl)global->batch_list->closure)->mode = 4;
}

```

- 413 -

```

    | RGB_YUV {
        Dprintf("Gram: RGB/YUV\n");
        ((CopyCtrl)global->batch_list->closure)->mode=5;
    }
    | GAMMA {
        Dprintf("Gram: Gamma convert\n");
        ((CopyCtrl)global->batch_list->closure)->mode=6;
    };

comp_arg : VIDEO_NAME string {
    Dprintf("Gram: Compress name %s\n", $2);

    strcpy(((CompCtrl)global->batch_list->closure)->name, $2);
}
    | STATS_NAME string {
        Dprintf("Gram: Stats name %s\n", $2);

        strcpy(((CompCtrl)global->batch_list->closure)->stats_name, $2);

        ((CompCtrl)global->batch_list->closure)->stats_switch=True;
    }
    | BIN_NAME string {
        Dprintf("Gram: Bin name %s\n", $2);

        strcpy(((CompCtrl)global->batch_list->closure)->bin_name, $2);

        ((CompCtrl)global->batch_list->closure)->bin_switch=True;
    }
    | STILL_MODE {
        Dprintf("Gram: Still\n");
        ((CompCtrl)global->batch_list->closure)->stillvid=True;
    }

```

- 414 -

```
| VIDEO_MODE {  
    Dprintf("Gram: Video\n");  
    ((CompCtrl)global->batch_list->closure)->stillvid=False;  
}  
| AUTO_Q boolean {  
    Dprintf("Gram: Auto_q %s\n", $2?"True":"False");  
    ((CompCtrl)global->batch_list->closure)->auto_q=$2;  
}  
| QUANT_CONST fnumber {  
    Dprintf("Gram: Quant const %f\n", $2);  
  
    ((CompCtrl)global->batch_list->closure)->quant_const=$2;  
}  
| THRESH_CONST fnumber {  
    Dprintf("Gram: Thresh const %f\n", $2);  
  
    ((CompCtrl)global->batch_list->closure)->thresh_const=$2;  
}  
| BASE_FACTOR number fnumber {  
    Dprintf("Gram: Base factor oct %d = %f\n", $2, $3);  
  
    ((CompCtrl)global->batch_list->closure)->base_factors[$2]=$3;  
}  
| DIAG_FACTOR fnumber {  
    Dprintf("Gram: Diag factor %f\n", $2);  
    ((CompCtrl)global->batch_list->closure)->diag_factor=$2;  
}  
| CHROME_FACTOR fnumber {  
    Dprintf("Gram: Chrome factor %f\n", $2);  
  
    ((CompCtrl)global->batch_list->closure)->chrome_factor=$2;  
}
```

- 415 -

```

| DECISION decision {
    Dprintf("Gram: Decision changed\n");
    ((CompCtrl)global->batch_list->closure)->decide=$2;
}
| FEEDBACK number {
    ((CompCtrl)global->batch_list->closure)->feedback=$2;
    ((CompCtrl)global->batch_list->closure)->auto_q=True;
}
| FILTER filter {
    String filters[2]={"None","Exp"};
    Dprintf("Gram: Filter %s\n",filters[$2]);
    ((CompCtrl)global->batch_list->closure)->filter=$2;
}
| CMP_CONST fnumber {
    Dprintf("Gram: Comparison %f\n",$2);
    ((CompCtrl)global->batch_list->closure)->cmp_const=$2;
}
| FPS fnumber {
    Dprintf("Gram: Frame Rate %f\n",$2);
    ((CompCtrl)global->batch_list->closure)->fps=$2;
}
| BITRATE number {
    Dprintf("Gram: %dx64k/s\n",$2);
    ((CompCtrl)global->batch_list->closure)->bitrate=$2;
}
| BUFFER {
    Dprintf("Gram: Buffer on\n");

    ((CompCtrl)global->batch_list->closure)->buf_switch=True;
};

decision      : DEC_MAX{ $$ = 0; }

```


- 416 -

```

| DEC_SIGABS { $$ = 1; }
| DEC_SIGSQR { $$ = 2; };

filter      : FLT_NONE { $$ = 0; }
              | FLT_EXP { $$ = 1; };

trans_arg   : VIDEO_NAME string {
              Dprintf("Gram: Transform name %s\n", $2);

strcpy((((TransCurl)global->batch_list->closure)->name, $2);
        }
        | DIRECTION boolean {
          Dprintf("Gram: Direction %s\n", $2?"True":"False");
          (((TransCurl)global->batch_list->closure)->dirn=$2;
        }
        | SPACE number number {
          Dprintf("Gram: Space %d %d\n", $2, $3);
          (((TransCurl)global->batch_list->closure)->space[0]=$2;
          (((TransCurl)global->batch_list->closure)->space[1]=$3;
        }
        | PRECISION number {
          Dprintf("Gram: Precision %d bits\n", 8+$2);
          (((TransCurl)global->batch_list->closure)->precision=$2;
        };

boolean     : BOOLEAN { $$ = $1; };

string : STRING {
        ptr = (char *)malloc(strlen($1)+1);
        strcpy(ptr, $1);
        ptr[strlen(ptr)-1] = '\0';
        $$ = ptr;

```

};

fnumber : FNUMBER { \$\$ = \$1; };

number : NUMBER { \$\$ = \$1; };

%%

yyerror(s) char *s; {

 Eprintf("Gram: error %s\n",s);

 exit(3);

}

void NewBatch(proc,closure,call_data)

Proc proc;

caddr_t closure, call_data;

{

 Batch bat=(Batch)MALLOC(sizeof(BatchRec));

 bat->proc=proc;

 bat->closure=closure;

 bat->call_data=call_data;

 bat->next=global->batch_list;

 global->batch_list=bat;

}

- 418 -

source/Klics.h

/* Block size - no not change */

#define BLOCK 2

typedef int Block[BLOCK][BLOCK]; /* small block */

/* tokens */

#define TOKENS 15

#define ZERO_STILL 0

#define NON_ZERO_STILL 1

#define BLOCK_SAME 2

#define ZERO_VID 3

#define BLOCK_CHANGE 4

#define LOCAL_ZERO 5

#define LOCAL_NON_ZERO 6

#define CHANNEL_ZERO 7

#define CHANNEL_NON_ZERO 8

#define OCT_ZERO 9

#define OCT_NON_ZERO 10

#define LPF_ZERO 11

#define LPF_NON_ZERO 12

#define LPF_LOC_ZERO 13

#define LPF_LOC_NON_ZERO 14

static int token_bits[TOKENS]

= {1,1,1,2,2,1,1,1,1,1,1,1,1,1,1};

static unsigned char token_codes[TOKENS] = {0,1,0,1,3,0,1,0,1,0,1,0,1,0,1};

- 419 -

/* decision algorithms */

#define MAXIMUM 0

#define SIGABS 1

#define SIGSQR 2

/* compression modes */

#define STILL 0

#define SEND 1

#define VOID 2

#define STOP 3

/* LookAhead histogram */

#define HISTO 400

#define HISTO_DELTA 20.0

#define HISTO_BITS 9

#include "../include/Bits.h"

typedef struct {

Video src, dst;

Boolean stillvid, stats_switch, bin_switch, auto_q, buf_switch;

double quant_const, thresh_const, cmp_const, fps,
base_factors[5], diag_factor, chrome_factor;

int bitrate, feedback, decide, filter;

char name[STRLEN], stats_name[STRLEN], bin_name[STRLEN],

src_name[STRLEN];

Bits bfp;

} CompCtrlRec, *CompCtrl;

typedef struct {

Boolean stillvid, auto_q, buf_switch;

double quant_const, thresh_const, cmp_const, fps,

- 420 -

```

        base_factors[5], diag_factor, chrome_factor;
int    decide;
VideoFormat type;
Boolean    disk, gamma;
int    rate, start, size[3], UVsample[2];
VideoTrans    trans;
int    precision;
} KlicsHeaderRec, *KlicsHeader;

```

- 421 -

source/KlicsSA.h

#include <stdio.h>

#include "Bits.h"

#define negif(bool,value) ((bool)?-(value):(value))

extern Bits bopen();

extern void bclose(), bread(), bwrite(), bflush();

/* Stand Alone definitions to replace VideoRec & CompCtrl assumes:

* video->type == YUV;

* video->UVsample[] = {1,1};

* video->trans.wavelet.space[] = {3,2};

* ctrl->bin_switch == True;

*/

#define SA_WIDTH 352

#define SA_HEIGHT 288

#define SA_PRECISION 2

static double base_factors[5] = {1.0,0.32,0.16,0.16,0.16};

#define diag_factor 1.4142136

#define chrome_factor 2.0

#define thresh_const 0.6

#define cmp_const 0.9

/* Block size - no not change */

#define BLOCK 2

typedef int Block[BLOCK][BLOCK]; /* small block */

- 422 -

/* tokens */

```
#define      TOKENS      15

#define ZERO_STILL      0
#define NON_ZERO_STILL  1
#define BLOCK_SAME      2
#define ZERO_VID        3
#define BLOCK_CHANGE    4
#define LOCAL_ZERO      5
#define LOCAL_NON_ZERO   6
#define CHANNEL_ZERO    7
#define CHANNEL_NON_ZERO 8
#define OCT_ZERO        9
#define OCT_NON_ZERO    10
#define LPF_ZERO        11
#define LPF_NON_ZERO    12
#define LPF_LOC_ZERO    13
#define LPF_LOC_NON_ZERO 14
```

```
static int          token_bits[TOKENS]
= {1,1,1,2,2,1,1,1,1,1,1,1,1,1,1};
static unsigned char token_codes[TOKENS] = {0,1,0,1,3,0,1,0,1,0,1,0,1,0,1};
```

/* decision algorithms */

```
#define MAXIMUM 0
#define SIGABS  1
#define SIGSQR  2
```

/* compression modes */

```
#define STILL  0
#define SEND   1
#define VOID   2
```

- 423 -

#define STOP 3

/* LookAhead histogram */

#define HISTO 400

#define HISTO_DELTA 20.0

#define HISTO_BITS 9

source/Lex.l

%{

/*

* Lex driver for input files: .pal .vid .bat

*/

#include "../include/xwave.h"

#include "../include/Gram.h"

extern int ParseInput();

#undef unput

#undef input

#undef output

#undef feof

#define unput(c) ungetc(c,global->parse_fp)

#define input() ParseInput(global->parse_fp)

#define output(c) putchar(c)

#define feof() (1)

%}

number -?[0-9]+

fnumber -?[0-9]+ "."[0-9]+

string \"([^\"]|\\.)*\"

%start WAIT MAP VIDEO BATCH BATCH_TRANS BATCH_COMP

%n 2000

%p 4000

%e 2000

- 425 -

%%

```
/* { char c='\0';
```

```
    while(c!='') {
        while (c!='') c=input();
        while (c=='') c=input();
    }
}
```

```
\.pal { BEGIN MAP; Dprintf("Lex: Reading palette file\n"); return(FILE_PAL); }
\.vid { BEGIN VIDEO; Dprintf("Lex: Reading video file\n"); return(FILE_VID); }
\.bat { BEGIN BATCH; Dprintf("Lex: Reading batch file\n"); return(FILE_BAT); }
```

```
{number}      { (void)sscanf(yytext, "%d", &yyval.num); return(NUMBER); }
{string}      { yyval.ptr = (char *)yytext; return(STRING); }
{fnumber}     { (void)sscanf(yytext, "%lf", &yyval.fnum); return(FNUMBER); }
```

```
<MAP> Palette { return(PALETTE); }
<MAP> \{      { return(LEFT_BRACE); }
<MAP> \}      { return(RIGHT_BRACE); }
<MAP> Range   { return(RANGE); }
<MAP> Line    { return(LINE); }
```

```
<VIDEO> Type   { return(TYPE); }
<VIDEO> MONO   { return(FORMAT_MONO); }
<VIDEO> RGB    { return(FORMAT_RGB); }
<VIDEO> YUV    { return(FORMAT_YUV); }
<VIDEO> Rate   { return(RATE); }
<VIDEO> Disk   { return(DISK); }
<VIDEO> Gamma  { return(GAMMA); }
<VIDEO> Negative { return(NEGATIVE); }
```

- 426 -

```
<VIDEO> Path          { return(PATH); }
<VIDEO> Files         { return(FILES); }
<VIDEO> Transform     { return(TRANSFORM); }
<VIDEO> None          { return(TRANSFORM_NONE); }
<VIDEO> Wavelet       { return(TRANSFORM_WAVE); }
<VIDEO> Start         { return(START); }
<VIDEO> End           { return(END); }
<VIDEO> Length        { return(LEN); }
<VIDEO> Dimensions    { return(DIM); }
<VIDEO> Header        { return(HEADER); }
<VIDEO> Offsets       { return(OFFSETS); }
<VIDEO> Size          { return(SIZE); }
<VIDEO> Precision     { return(PRECISION); }
<VIDEO> Yes           { yy1val.bool=True; return(BOOLEAN); }
<VIDEO> No            { yy1val.bool=False; return(BOOLEAN); }

<BATCH> Load          { return(LOAD); }
<BATCH> Save           { return(SAVE); }
<BATCH> SaveAbekus     { return(SAVE_ABEKUS); }
<BATCH> Compare        { return(COMPARE); }
<BATCH> Drop           { return(DROP); }
<BATCH> ImportKLICS    { return(IMPORT_KLICS); }
<BATCH> Transform      { BEGIN BATCH_TRANS; return(TRANSFORM); }
<BATCH> Compress       { BEGIN BATCH_COMP; return(COMPRESS); }
<BATCH> Xwave          { return(XWAVE); }
<BATCH> Shell          { return(SHELL); }
<BATCH> Copy           { return(COPY); }
<BATCH> Direct         { return(DIRECT_COPY); }
<BATCH> Diff           { return(DIFF); }
<BATCH> LPFzero        { return(LPF_WIPE); }
<BATCH> LPFonly        { return(LPF_ONLY); }
<BATCH> RGB-YUV        { return(RGB_YUV); }
```

- 427 -

```

<BATCH> Gamma      { return(GAMMA); }

<BATCH_COMP> VideoName { return(VIDEO_NAME); }
<BATCH_COMP> Stats     { return(STATS_NAME); }
<BATCH_COMP> Binary    { return(BIN_NAME); }
<BATCH_COMP> Yes       { yylval.bool=True; return(BOOLEAN); }
<BATCH_COMP> No        { yylval.bool=False; return(BOOLEAN); }
<BATCH_COMP> Still     { return(STILL_MODE); }
<BATCH_COMP> Video     { return(VIDEO_MODE); }
<BATCH_COMP> AutoQuant { return(AUTO_Q); }
<BATCH_COMP> QuantConst { return(QUANT_CONST); }
<BATCH_COMP> ThreshConst { return(THRESH_CONST); }
<BATCH_COMP> BaseFactor { return(BASE_FACTOR); }
<BATCH_COMP> DiagFactor { return(DIAG_FACTOR); }
<BATCH_COMP> ChromeFactor { return(CHROME_FACTOR); }
<BATCH_COMP> Decision  { return(DECISION); }
<BATCH_COMP> Feedback  { return(FEEDBACK); }
<BATCH_COMP> Maximum   { return(DEC_MAX); }
<BATCH_COMP> SigmaAbs   { return(DEC_SIGABS); }
<BATCH_COMP> SigmaSqr   { return(DEC_SIGSQR); }
<BATCH_COMP> Filter     { return(FILTER); }
<BATCH_COMP> None       { return(FLT_NONE); }
<BATCH_COMP> Exp        { return(FLT_EXP); }
<BATCH_COMP> CmpConst   { return(CMP_CONST); }
<BATCH_COMP> FrameRate  { return(FPS); }
<BATCH_COMP> Bitrate    { return(BITRATE); }
<BATCH_COMP> Buffer      { return(BUFFER); }
<BATCH_COMP> \{         { return(LEFT_BRACE); }
<BATCH_COMP> \}         { END; BEGIN BATCH;
return(RIGHT_BRACE); }

<BATCH_TRANS> VideoName { return(VIDEO_NAME); }

```

```

< BATCH_TRANS > Direction    { return(DIRECTION); }
< BATCH_TRANS > Space { return(SPACE); }
< BATCH_TRANS > Precision    { return(PRECISION); }
< BATCH_TRANS > Yes          { yy1val.bool=True; return(BOOLEAN); }
< BATCH_TRANS > No           { yy1val.bool=False; return(BOOLEAN); }
< BATCH_TRANS > \{           { return(LEFT_BRACE); }
< BATCH_TRANS > \}           { END; BEGIN BATCH; return(RIGHT_BRACE); }

```

```

[. \t\n]          { ; }

```

%%

```

yywrap() { return(1); }

```

- 429 -

source/Transform.h

```
typedef struct {  
    Video src;  
    char name[STRLEN], src_name[STRLEN];  
    int space[2], precision;  
    Boolean dirn;  
} TransCtrlRec, *TransCtrl;
```

- 430 -

source/Video.h

```
typedef struct {  
    char names[4][STRLEN];  
} AbekusCtrlRec, *AbekusCtrl;
```

- 431 -

source/makefile

Xwave Makefile

#

CFLAGS = -O -I../include

LIBS = -lXaw -lXmu -lXt -lXext -lX11 -lm -ll -L/usr/openwin/lib

.KEEP_STATE:

.SUFFIXES: .c .o

xwaveSRC = Select.c Convert.c xwave.c InitMain.c Pop2.c Video2.c Malloc.c
InitFrame.c \

Frame.c Transform.c Convolve3.c Update.c Image.c Menu.c

PullRightMenu.c \

NameButton.c SmeBSBpr.c Process.c Lex.c Gram.c Parse.c Color.c \

Bits.c Storage.c Copy.c Message.c Palette.c ImportKlics.c Icon3.c Klics5.c

\

KlicsSA.c KlicsTestSA.c ImportKlicsSA.c ImpKlicsTestSA.c

objDIR = ../\$(ARCH)

xwaveOBJ = \$(xwaveSRC:%.c=\$(objDIR)/%.o)

\$(objDIR)/xwave: \$(xwaveOBJ)

gcc -o \$@ \$(xwaveOBJ) \$(LIBS) \$(CFLAGS)

echo

\$(xwaveOBJ): \$\$(@F:.o=.c) ../include/xwave.h

gcc -c \$\$(@F:.o=.c) \$(CFLAGS) -o \$@

Lex.c: Gram.c Lex.l

- 432 -

```
lex Lex.l
```

```
mv lex.yy.c Lex.c
```

```
Gram.c: Gram.y
```

```
bison -dlt Gram.y
```

```
mv $(@F:.c=.tab.h) ../include/Gram.h
```

```
mv $(@F:.c=.tab.c) Gram.c
```

include/Bits.h

#ifndef _Bits_h

#define _Bits_h

```
typedef struct {
    unsigned char buf;
    int bufsize;
    FILE *fp;
} BitsRec, *Bits;
```

#endif

include/DTheader.h

```
typedef struct DTheader {
    char file_id[8];          /* "DT-IMAGE" */
    char struct_id;           /* 1 */
    char prod_id;             /* 4 */
    char util_id;             /* 1 */
    char board_id;            /* 2 */
    char create_time[9]; /* [0-1]year, [2]month, [3]dayofmonth, [4]dayofweek,
[5]hour, [6]min, [7]sec, [8]sec/100 */
    char mod_time[9];         /* as create_time */
    char datum;               /* 1 */
    char datasize[4];         /* 1024?? */
    char file_struct;         /* 1 */
    char datatype;            /* 1 */
    char compress;            /* 0 */
    char store;               /* 1 */
    char aspect[2];           /* 4, 3 */
    char bpp;                 /* 8 */
    char spatial;             /* 1 */
    char width[2];            /* 512 */
    char height[2];           /* 512 */
    char full_width[2];       /* 512 */
    char full_height[2];      /* 512 */
    char unused1[45];
    char comment[160];
    char unused2[256];
} DTheader;
```

include/icon.h

```
typedef      enum {  
    FW_label, FW_icon, FW_command, FW_text, FW_buton, FW_icon_buton,  
    FW_view, FW_toggle,  
    FW_yn,  
    FW_up, FW_down, FW_integer,  
    FW_scroll, FW_float,  
    FW_form,  
} FormWidgetType;
```

```
typedef      enum {  
    SW_below, SW_over, SW_top, SW_menu,  
} ShellWidgetType;
```

```
typedef      struct {  
    String name;  
    String contents;  
    int      fromHoriz, fromVert;  
    FormWidgetType type;  
    String hook;  
} FormItem;
```

- 436 -

include/Image.h

/*

* \$XConsortium: Image.h, v 1.24 89/07/21 01:48:51 kit Exp \$

*/

/*****

Copyright 1987, 1988 by Digital Equipment Corporation, Maynard, Massachusetts,
and the Massachusetts Institute of Technology, Cambridge, Massachusetts.

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in
supporting documentation, and that the names of Digital or MIT not be
used in advertising or publicity pertaining to distribution of the
software without specific, written prior permission.

DIGITAL DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
INCLUDING

ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO
EVENT SHALL

DIGITAL BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL
DAMAGES OR

ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR
PROFITS,

WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
TORTIOUS ACTION,

ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF
THIS
SOFTWARE.

```
...../
1

#ifndef _XawImage_h
#define _XawImage_h
```

```
...../
*
* Image Widget
*
...../
```

```
#include <X11/Xaw/Simple.h>
#include <X11/Xmu/Converters.h>
```

```
/* Resources:
```

Name	Class	RepType	Default Value
border	BorderColor	Pixel	XtDefaultForeground
borderWidth	BorderWidth	Dimension	1
cursor	Cursor	Cursor	None
destroyCallback	Callback	XtCallbackList	NULL
insensitiveBorder	Insensitive	Pixmap	Gray
mappedWhenManaged	MappedWhenManaged	Boolean	True
sensitive	Sensitive	Boolean	True
bitmap	Bitmap	Pixmap	NULL
callback	Callback	XtCallbackList	NULL
x	Position	Position	0

- 438 -

y Position Position 0

*/

#define XtNbitmap "bitmap"

#define XtCBitmap "Bitmap"

/* Class record constants */

extern WidgetClass imageWidgetClass;

typedef struct _ImageClassRec *ImageWidgetClass;

typedef struct _ImageRec *ImageWidget;

#endif /* _XawImage_h */

/* DON'T ADD STUFF AFTER THIS #endif */

- 439 -

include/ImageHeader.h

```

/* Author: Philip R. Thompson
 * Address: phils@athena.mit.edu, 9-526
 * Note: size of header should be 1024 (1K) bytes.
 * $Header: ImageHeader.h,v 1.2 89/02/13 09:01:36 phils Locked $
 * $Date: 89/02/13 09:01:36 $
 * $Source: /mit/phils/utis/RCS/ImageHeader.h,v $
 */

#define IMAGE_VERSION 3

typedef struct ImageHeader {
    char file_version[8]; /* header version */
    char header_size[8]; /* Size of file header in bytes */
    char image_width[8]; /* Width of the raster image */
    char image_height[8]; /* Height of the raster image */
    char num_colors[8]; /* Actual number of entries in c_map */
    char num_channels[8]; /* 0 or 1 = pixmap, 3 = RG&B buffers */
    char num_pictures[8]; /* Number of pictures in file */
    char alpha_channel[4]; /* Alpha channel flag */
    char runlength[4]; /* Runlength encoded flag */
    char author[48]; /* Name of who made it */
    char date[32]; /* Date and time image was made */
    char program[16]; /* Program that created this file */
    char comment[96]; /* other viewing info. for this image */
    unsigned char c_map[256][3]; /* RGB values of the pixmap indices */
} ImageHeader;

```

/* Note:

* - All data is in char's in order to maintain easily portability

- 440 -

- * across machines and some human readability.
- * - Images may be stored as pixmaps or in seperate channels, such as
- * red, green, blue data.
- * - An optional alpha channel is seperate and is found after every
- * num_channels of data.
- * - Pixmaps, red, green, blue, alpha and other channel data are stored
- * sequentially after the header.
- * - If num_channels = 1 or 0, a pixmap is assumed and up to num_colors
- * of colormap in the header are used.
- */

/***/ end ImageHeader.h ***/

- 441 -

include/ImageP.h

/*

* \$XConsortium: ImageP.h,v 1.24 89/06/08 18:05:01 swick Exp \$

*/

/******

Copyright 1987, 1988 by Digital Equipment Corporation, Maynard, Massachusetts,
and the Massachusetts Institute of Technology, Cambridge, Massachusetts.

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in
supporting documentation, and that the names of Digital or MIT not be
used in advertising or publicity pertaining to distribution of the
software without specific, written prior permission.

DIGITAL DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
INCLUDING

ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO
EVENT SHALL

DIGITAL BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL
DAMAGES OR

ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR
PROFITS,

WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER

- 442 -

TORTIOUS ACTION,
ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF
THIS
SOFTWARE.

```
.....*/

/*
 * ImageP.h - Private definitions for Image widget
 *
 */

#ifndef _XawImageP_h
#define _XawImageP_h

/*****
 *
 * Image Widget Private Data
 *
 *****/

#include "../include/Image.h"
#include <X11/Xaw/SimpleP.h>

/* New fields for the Image widget class record */

typedef struct {int foo;} ImageClassPart;

/* Full class record declaration */
typedef struct _ImageClassRec {
    CoreClassPart    core_class;
    SimpleClassPart  simple_class;
```

- 443 -

```

    ImageClassPart  image_class;
} ImageClassRec;

```

```

extern ImageClassRec imageClassRec;

```

```

/* New fields for the Image widget record */

```

```

typedef struct {

```

```

    /* resources */

```

```

        Pixmap      pixmap;

```

```

        XtCallbackList  callbacks;

```

```

    /* private state */

```

```

        Dimension    map_width, map_height;

```

```

} ImagePart;

```

```

/*****

```

```

 *

```

```

 * Full instance record declaration

```

```

 *

```

```

*****/

```

```

typedef struct _ImageRec {

```

```

    CorePart  core;

```

```

    SimplePart  simple;

```

```

    ImagePart  image;

```

```

} ImageRec;

```

```

#endif /* _XawImageP_h */

```

- 444 -

include/Message.h

```
typedef struct {
    Widget      shell, widget; /* shell and text widgets (NULL if not created */
    XawTextBlock info; /* Display text */
    int         size, rows, cols; /* Size of buffer (info.ptr) & dimensions of display */
    XawTextEditType edit; /* edit type */
    Boolean     own_text; /* text is owned by message? */
} MessageRec, *Message;
```

- 445 -

include/Palette.h

```
#define PalettePath "."  
#define   PaletteExt  ".pal"
```

```
typedef      struct _MapRec {  
    int      start, finish, m, c;  
    struct _MapRec  *next;  
} MapRec, *Map;
```

```
typedef      struct _PaletteRec {  
    char      name[STRLEN];  
    Map      mappings;  
    struct _PaletteRec  *next;  
} PaletteRec, *Palette;
```

- 446 -

include/PullRightMenu.h

/*

* \$XConsortium: PullRightMenu.h,v 1.17 89/12/11 15:01:55 kit Exp \$

*

* Copyright 1989 Massachusetts Institute of Technology

*

* Permission to use, copy, modify, distribute, and sell this software and its
* documentation for any purpose is hereby granted without fee, provided that
* the above copyright notice appear in all copies and that both that
* copyright notice and this permission notice appear in supporting
* documentation, and that the name of M.I.T. not be used in advertising or
* publicity pertaining to distribution of the software without specific,
* written prior permission. M.I.T. makes no representations about the
* suitability of this software for any purpose. It is provided "as is"
* without express or implied warranty.

*

* M.I.T. DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE.
INCLUDING ALL

* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO
EVENT SHALL M.I.T.

* BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES
OR ANY DAMAGES

* WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION

* OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT
OF OR IN

* CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

*

*/

- 447 -

/*

* PullRightMenu.h - Public Header file for PullRightMenu widget.

*

* This is the public header file for the Athena PullRightMenu widget.

* It is intended to provide one pane pulldown and popup menus within

* the framework of the X Toolkit. As the name implies it is a first and

* by no means complete implementation of menu code. It does not attempt to

* fill the needs of all applications, but does allow a resource oriented

* interface to menus.

*

*/

#ifndef _PullRightMenu_h

#define _PullRightMenu_h

#include <X11/Shell.h>

#include <X11/Xmu/Converters.h>

/*****

*

* PullRightMenu widget

*

*****/

/* PullRightMenu Resources:

Name	Class	RepType	Default Value
background	Background	Pixel	XtDefaultBackground
backgroundPixmap	BackgroundPixmap	Pixmap	None
borderColor	BorderColor	Pixel	XtDefaultForeground
borderPixmap	BorderPixmap	Pixmap	None

- 448 -

borderWidth	BorderWidth	Dimension	1
bottomMargin	VerticalMargins	Dimension	VerticalSpace
columnWidth	ColumnWidth	Dimension	Width of widest text
cursor	Cursor	Cursor	None
destroyCallback	Callback	Pointer	NULL
height	Height	Dimension	0
label	Label	String	NULL (No label)
labelClass	LabelClass	Pointer	smeBSBObjectClass
mappedWhenManaged	MappedWhenManaged	Boolean	True
rowHeight	RowHeight	Dimension	Height of Font
sensitive	Sensitive	Boolean	True
topMargin	VerticalMargins	Dimension	VerticalSpace
width	Width	Dimension	0
button	Widget	Widget	NULL
x	Position	Position	0
y	Position	Position	0

*/

```
typedef struct _PullRightMenuClassRec* PullRightMenuWidgetClass;
typedef struct _PullRightMenuRec* PullRightMenuWidget;
```

```
extern WidgetClass pullRightMenuWidgetClass;
```

```
#define XtNcursor "cursor"
#define XtNbottomMargin "bottomMargin"
#define XtNcolumnWidth "columnWidth"
#define XtNlabelClass "labelClass"
#define XtNmenuOnScreen "menuOnScreen"
#define XtNpopupOnEntry "popupOnEntry"
#define XtNrowHeight "rowHeight"
#define XtNtopMargin "topMargin"
```

- 449 -

```
#define XtNbutton    "button"
```

```
#define XtCColumnWidth "ColumnWidth"
```

```
#define XtCLabelClass "LabelClass"
```

```
#define XtCMenuOnScreen "MenuOnScreen"
```

```
#define XtCPopupOnEntry "PopupOnEntry"
```

```
#define XtCRowHeight "RowHeight"
```

```
#define XtCVerticalMargins "VerticalMargins"
```

```
#define      XtCWidget    "Widget"
```

```
/******
```

```
*
```

```
* Public Functions.
```

```
*
```

```
*****/
```

```
/*  Function Name: XawPullRightMenuAddGlobalActions
```

```
*  Description: adds the global actions to the simple menu widget.
```

```
*  Arguments: app_con - the appcontext.
```

```
*  Returns: none.
```

```
*/
```

```
void
```

```
XawpullRightMenuAddGlobalActions(/* app_con */);
```

```
/*
```

```
XtAppContext app_con;
```

```
*/
```

```
#endif /* _PullRightMenu_h */
```

include/SmeBSBpr.h

/*

* SXConsortium: SmeBSB.h,v 1.5 89/12/11 15:20:14 kit Exp \$

*

* Copyright 1989 Massachusetts Institute of Technology

*

* Permission to use, copy, modify, distribute, and sell this software and its
 * documentation for any purpose is hereby granted without fee, provided that
 * the above copyright notice appear in all copies and that both that
 * copyright notice and this permission notice appear in supporting
 * documentation, and that the name of M.I.T. not be used in advertising or
 * publicity pertaining to distribution of the software without specific,
 * written prior permission. M.I.T. makes no representations about the
 * suitability of this software for any purpose. It is provided "as is"
 * without express or implied warranty.

*

* M.I.T. DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
 INCLUDING ALL

* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO
 EVENT SHALL M.I.T.

* BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES
 OR ANY DAMAGES

* WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
 WHETHER IN AN ACTION

* OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT
 OF OR IN

* CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

*/

- 451 -

```

/*
 * SmeBSBpr.h - Public Header file for SmeBSB object.
 *
 * This is the public header file for the Athena BSB Sme object.
 * It is intended to be used with the simple menu widget. This object
 * provides bitmap - string - bitmap style entries.
 *
 */

```

```

#ifndef _SmeBSBpr_h

```

```

#define _SmeBSBpr_h

```

```

#include <X11/Xmu/Converters.h>

```

```

#include <X11/Xaw/Sme.h>

```

```

/*****

```

```

 *
 * SmeBSBpr object
 *

```

```

*****/

```

```

/* BSB pull-right Menu Entry Resources:

```

Name	Class	RepType	Default Value
callback	Callback	Callback	NULL
destroyCallback	Callback	Pointer	NULL
font	Font	XFontStruct *	XtDefaultFont
foreground	Foreground	Pixel	XtDefaultForeground
height	Height	Dimension	0
label	Label	String	Name of entry

- 452 -

leftBitmap	LeftBitmap	Pixmap	None
leftMargin	HorizontalMargins	Dimension	4
rightBitmap	RightBitmap	Pixmap	None
rightMargin	HorizontalMargins	Dimension	4
sensitive	Sensitive	Boolean	True
vertSpace	VertSpace	int	25
width	Width	Dimension	0
x	Position	Position	0n
y	Position	Position	0
menuName	MenuName	String	"menu"

*/

```
typedef struct _SmeBSBprClassRec    *SmeBSBprObjectClass;
```

```
typedef struct _SmeBSBprRec        *SmeBSBprObject;
```

```
extern WidgetClass smeBSBprObjectClass;
```

```
#define XtNleftBitmap "leftBitmap"
```

```
#define XtNleftMargin "leftMargin"
```

```
#define XtNrightBitmap "rightBitmap"
```

```
#define XtNrightMargin "rightMargin"
```

```
#define XtNvertSpace "vertSpace"
```

```
#define XtNmenuName "menuName"
```

```
#define XtCLeftBitmap "LeftBitmap"
```

```
#define XtCHorizontalMargins "HorizontalMargins"
```

```
#define XtCRightBitmap "RightBitmap"
```

```
#define XtCVertSpace "VertSpace"
```

```
#define XtCMenuName "MenuName"
```

```
#endif /* _SmeBSBpr_h */
```

- 453 -

include/SmeBSBP.h

/*

- * \$XConsortium: SmeBSBP.h,v 1.6 89/12/11 15:20:15 kit Exp \$

*

- * Copyright 1989 Massachusetts Institute of Technology

*

- * Permission to use, copy, modify, distribute, and sell this software and its
- * documentation for any purpose is hereby granted without fee, provided that
- * the above copyright notice appear in all copies and that both that
- * copyright notice and this permission notice appear in supporting
- * documentation, and that the name of M.I.T. not be used in advertising or
- * publicity pertaining to distribution of the software without specific,
- * written prior permission. M.I.T. makes no representations about the
- * suitability of this software for any purpose. It is provided "as is"
- * without express or implied warranty.

*

- * M.I.T. DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,

INCLUDING ALL

- * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO

EVENT SHALL M.I.T.

- * BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES

OR ANY DAMAGES

- * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,

WHETHER IN AN ACTION

- * OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT

OF OR IN

- * CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

*

- * Author: Chris D. Peterson, MIT X Consortium

- 454 -

*/

/*

* SmeP.h - Private definitions for Sme object

*

*/

#ifndef _XawSmeBSBP_h

#define _XawSmeBSBP_h

/*****

*

* Sme Object Private Data

*

*****/

#include <X11/Xaw/SmeP.h>

#include "../include/SmeBSBpr.h"

/*****

*

* New fields for the Sme Object class record.

*

*****/

typedef struct _SmeBSBprClassPart {

XtPointer extension;

} SmeBSBprClassPart;

/* Full class record declaration */

typedef struct _SmeBSBprClassRec {

RectObjClassPart rect_class;

- 455 -

```

SmeClassPart    sme_class;
SmeBSBprClassPart sme_bsb_class;
} SmeBSBprClassRec;

```

```

extern SmeBSBprClassRec smeBSBprClassRec;

```

```

/* New fields for the Sme Object record */

```

```

typedef struct {
    /* resources */
    String label;          /* The entry label. */
    int vert_space;        /* extra vert space to leave, as a percentage
                           of the font height of the label. */
    Pixmap left_bitmap, right_bitmap; /* bitmaps to show. */
    Dimension left_margin, right_margin; /* left and right margins. */
    Pixel foreground;      /* foreground color. */
    XFontStruct * font;    /* The font to show label in. */
    XtJustify justify;     /* Justification for the label. */
    String menu_name;      /* Popup menu name */

```

```

/* private resources. */

```

```

Boolean set_values_area_cleared; /* Remember if we need to unhighlight. */
GC norm_gc;                      /* normal color gc. */
GC rev_gc;                       /* reverse color gc. */
GC norm_gray_gc;                 /* Normal color (grayed out) gc. */
GC invert_gc;                    /* gc for flipping colors. */

```

```

Dimension left_bitmap_width; /* size of each bitmap. */
Dimension left_bitmap_height;
Dimension right_bitmap_width;
Dimension right_bitmap_height;

```


- 456 -

```
} SmeBSBprPart;
```

```
/* .....  
*  
* Full instance record declaration  
*  
* .....*/
```

```
typedef struct _SmeBSBprRec {  
    ObjectPart    object;  
    RectObjPart   rectangle;  
    SmePart       sme;  
    SmeBSBprPart  sme_bsb;  
} SmeBSBprRec;
```

```
/* .....  
*  
* Private declarations.  
*  
* .....*/
```

```
#endif /* _XawSmeBSBPpr_h */
```

include/xwave.h

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xatom.h>
#include <X11/Xaw/Cardinals.h>
#include <X11/StringDefs.h>
#include <X11/Xmu/Xmu.h>
#include <X11/Xaw/Command.h>
#include <X11/Xaw/List.h>
#include <X11/Xaw/Box.h>
#include <X11/Xaw/Form.h>
#include <X11/Xaw/Scrollbar.h>
#include <X11/Xaw/Viewport.h>
#include <X11/Xaw/AsciiText.h>
#include <X11/Xaw/Dialog.h>
#include <X11/Xaw/MenuButton.h>
#include <X11/Xaw/SimpleMenu.h>
#include <X11/Xaw/SmeBSB.h>
#include <X11/Xaw/Toggle.h>
#include "SmeBSBpr.h"
#include "PullRightMenu.h"
#include <X11/Shell.h>
#include <X11/cursorfont.h>
#define STRLEN 100
#define NAME_LEN 20
#include "Image.h"
#include "Message.h"
#include <dirent.h>
#include <math.h>
```

- 458 -

```
#include    <stdio.h>
#include    "Palette.h"
#include    "Icon.h"

#define     PLOT_DIR    "graphs"
#define     PLOT_EXT    ".plot"
#define     ELLA_IN_DIR    "."
#define     ELLA_IN_EXT    ".eli"
#define     ELLA_OUT_DIR    "."
#define     ELLA_OUT_EXT    ".elo"
#define     VID_DIR    "videos"
#define     VID_EXT    ".vid"
#define     IMAGE_DIR    "images"
#define     BATCH_DIR    "batch"
#define     BATCH_EXT    ".bat"
#define     KLICS_DIR    "import"
#define     KLICS_EXT    ".klics"
#define     KLICS_SA_DIR    "import"
#define     KLICS_SA_EXT    ".klicsSA"

typedef enum {
    TRANS_None, TRANS_Wave,
} TransType;

typedef     enum {
    MONO, RGB, YUV,
} VideoFormat;

extern String ChannelName[3][4];

#define     negif(bool,value)    ((bool)?-(value):(value))
```

- 459 -

```
typedef struct {
    String name;
    Pixmap pixmap;
    unsigned int height, width;
} IconRec, *Icon;
```

```
typedef void (*Proc)();
typedef String (*ListProc)();
typedef Boolean (*BoolProc)();
```

```
typedef struct {
    String name;
    WidgetClass widgetClass;
    String label;
    String hook; /* menuName for smeBSBprObjectClass */
} MenuItem;
```

```
typedef struct {
    String name, button;
    ListProc list_proc;
    String action_name;
    Proc action_proc;
    caddr_t action_closure;
} SelectItem, *Selection;
```

```
typedef struct {
    TransType type;
    int space[2];
    Boolean dirn;
} WaveletTrans;
```

```
typedef union {
```

- 460 -

```

TransType    type;
WaveletTrans  wavelet;
} VideoTrans;

```

```

typedef      struct _VideoRec  {
    char  name[STRLEN];           /* Name of this video name.vid */
    char  path[STRLEN];           /* Path to frame file(s) */
    char  files[STRLEN];          /* Name of frames files001 if not name */
    VideoFormat type;             /* Type of video (MONO,RGB,YUV) */
    Boolean disk; /* Frames reside on disk rather than in memory */
    Boolean gamma;                /* Gamma corrected flag */
    Boolean negative;             /* Load negative values in data */
    int    rate;                  /* Frames per second */
    int    start;                 /* Starting frame number */
    int    size[3]; /* Dimensions of video after extraction x, y and z */
    int    UVsample[2];           /* Chrominance sub-sampling x and y */
    int    offset;                /* Header length */
    int    cols, rows;            /* Dimensions of video as stored */
    int    x_offset, y_offset; /* Offset of extracted video in stored */
    VideoTrans trans;             /* Transform technique used */
    int    precision;             /* Storage precision above 8 bits */
    short  **data[3];             /* Image data channels */
    struct _VideoRec *next;       /* Next video in list */
} VideoRec, *Video;

```

```

typedef      struct {
    Video video;
    char  name[STRLEN];
} VideoCtrlRec, *VideoCtrl;

```

```

typedef      struct _PointRec  {
    int    location[2];

```

- 461 -

```
int    usage;
struct _PointRec  *next;
} PointRec, *Point;

typedef struct _FrameRec {
    Widget      shell, image_widget, point_merge_widget;
    Video video;
    int    zoom, frame, channel, palette;
    Boolean    point_switch, point_merge;
    Point point;
    Message    msg;
    struct _FrameRec  *next;
} FrameRec, *Frame;

#define    NO_CMAPS 6

typedef struct _BatchRec {
    Proc    proc;
    caddr_t    closure, call_data;
    struct _BatchRec  *next;
} BatchRec, *Batch;

typedef struct {
    char    home[STRLEN];
    XtAppContext    app_con;
    Widget    toplevel;
    int    no_icons;
    Icon    icons;
    Video    videos;
    Frame    frames;
    Point    points;
    Palette    palettes;
```

- 462 -

```
int    no_pals;
String parse_file;
String parse_token;
FILE   *parse_fp;
XVisualInfo *visinfo;
int    levels, rgb_levels, yuv_levels[3];
Colormap  cmap[NO_CMAPS];
String batch;
Batch batch_list;
Boolean  debug;
int    dither[16][16];
} GlobalRec, *Global;
```

```
typedef struct {
    Widget    widgets[3];
    int    max, min, *value;
    String format;
} NumInputRec, *NumInput;
```

```
typedef struct {
    Widget    widgets[2];
    double    max, min, *value;
    String format;
} FloatInputRec, *FloatInput;
```

```
extern Global    global;
```

```
/* InitFrame.c */
```

```
extern Video FindVideo();
```

```
/* Pop2.c */
```

- 463 -

```
extern void  NAO;  
extern Widget  FindWidget();  
extern void  Destroy();  
extern void  Free();
```

```
/* Storage.c */
```

```
extern void  NewFrame();  
extern void  GetFrame();  
extern void  SaveFrame();  
extern void  FreeFrame();  
extern void  SaveHeader();  
extern Video CopyHeader();
```

```
/* Message.c */
```

```
extern void  TextSize();  
extern Message  NewMessage();  
extern void  MessageWindow();  
extern void  CloseMessage();  
extern void  Mprintf();  
extern void  Dprintf();  
extern void  Eprintf();  
extern void  Mflush();
```

```
/* Icon3.c */
```

```
extern void  FillForm();  
extern void  FillMenu();  
extern Widget  ShellWidget();  
extern Widget  FormatWidget();  
extern void  SimpleMenu();
```


- 464 -

```
extern int   TextWidth();
extern Icon  FindIcon();
extern void  NumIncDec();
extern void  FloatIncDec();
extern void  ChangeYN();
extern XFontStruct *FindFont();
```

DATA COMPRESSION AND DECOMPRESSION
GREGORY KNOWLES AND ADRIAN S. LEWIS

M-2357 US

APPENDIX B-1

```
MAC_ADDR_COUNTER_COL = (bool:ck, l_reset:reset, STRING[xsize]bit:block_cnt_length)
```

```
-->
```

```
(l_col,bool):
```

```
BEGIN
```

```
MAKE BASE_COUNTER_COL:base_counter_col.
```

```
JOIN (ck,reset:block_cnt_length) -->base_counter_col.
```

```
OUTPUT (base_counter_col[1], CASE base_counter_col[2]
```

```
OF count_carry:1
```

```
ELSE 1
```

```
ESAC)
```

```
END.
```

```
MAC_ADDR_COUNTER_ROW = (bool:ck, l_reset:reset, STRING[ysize]bit:block_cnt_length,bool:col_carry)
```

```
-->
```

```
(l_row,bool):
```

```
BEGIN
```

```
MAKE BASE_COUNTER_ROW:base_counter_row.
```

```
JOIN (ck,reset,col_carry:block_cnt_length,CASE col_carry #type conversion#
```

```
OF 1:count_carry
```

```
ELSE count_rst
```

```
ESAC) -->base_counter_row.
```

```
OUTPUT (base_counter_row[1], CASE base_counter_row[2]
```

```
OF count_carry:1
```

```
ELSE 1
```

```
ESAC)
```

```
END.
```

```
#the string base address calculators#
```

- 467 -

```
MAC NOMULT_MAC_READ = (bool:ck, !_reset:reset, bool:col_end, !_mux4:mux_control, STRING[17]bit:incr,
  STRING[17]bit:oct_add_factor, STRING[19]bit:base_u base_v)
```

```
->
  STRING[19]bit:
```

```
BEGIN
```

```
MAKE ADD_US_ACTEL[19,17]:add,
```

```
  MUX_2[STRING[17]bit]:mux.
```

```
LET
```

```
  next_addr = MUX_4[STRING[19]bit](add[2..20], ZERO[19]b'0', base_u, base_v, mux_control),
  dff = DFF_NO_LOAD[STRING[19]bit](ck, reset, next_addr, b'000000000000000000000000').
```

```
JOIN (dff, mux, b'1') -> add,
  (incr, oct_add_factor, CASE col_end
    OF t:right
    ELSE left
    ESAC) -> mux.
```

```
OUTPUT dff
END.
```

```
MAC S_SPA = (STRING[19]bit:in)
```

```
->
```

```
  (flag, !_sparc_addr):BIOP TRANSFORM_US.
  MAC SPA_S = (!_sparc_addr:in)
```

```
->
```

```
  (flag, STRING[19]bit):BIOP TRANSFORM_US.
```

```
MAC SPARC_ADDR = (bool:ck, !_reset:reset, bool:col_end, !_mux4:mux_control, [2]!_sparc_addr:oct_add_factor,
```

```

STRING[19]bit base_u base_v)
->                                t_sparc_addr:

BEGIN
LET out=NOMULT_MAC_READ(ck,reset,col_end,mux_control,(SPA_S oct_add_factor[1])[2][3..19],
    (SPA_S oct_add_factor[2])[2][3..19],base_u,base_v).
OUTPUT (S_SPA out)[2]
END.

#-----#

#the read and write address generator,input the initial image & block sizes for oct/0 at that channel#
FN ADDR_GEN_NOSCRATCH= (bool:ck,t_reset:reset,t_direction:direction,t_channel:channel,
    STRING[9]bit x_p_1,STRING[11]bit x3_p_1,STRING[12]bit x7_p_1,
    STRING [ysize]bit:oclave_row_length,STRING [xsize]bit:oclave_col_length,t_reset:oclave_reset,
    t_oclave:oclave,bool:y_done,bool:yv_done,t_load:oclave_finished,STRING [19]bit:base_u base_v)

->
((!_input_mux,t_sparcport,t_dwtpport#dwt#),t_load#IDWT data valid#,t_load#read_valid#
    ,t_count_control#row read col read#,(t_col,t_count_control)#addr_col_read#):
#the current oclave and when the block finishes the 3 oclave transform#
BEGIN

# ADDR_COUNTER_ROW:addr_row_write,#
# ADDR_COUNTER_COL:addr_col_write,#
MAKE ROW_COUNT_CARRY:addr_row_read,
    COL_COUNT:      addr_col_read,
    SPARC_ADDR:write addr_read_addr,
    MEM_CONTROL_NOSCRATCH:mem_control,

```

JKFF:zero_hh_bool read_done_bool. #write begins #

```

LET mem_sel
    = CASE octave
    OF ocl/0:uno,
       ocl/1:dos,
       ocl/2:tres,
       ocl/3:quatro
    ESAC,

    sparc_add_1 = MUX_4(! sparc_add){
        (addr/1),
        (addr/2),
        (addr/4),
        (addr/8),
        mem_sel},

    sparc_add_2_y = MUX_4(STRING[12]bit)(
        (b"00000000000001"),
        (b"000" CONC x_p_1[1..7] CONC b"10"),
        (b"0" CONC x3_p_1[1..8] CONC b"100"),
        (x7_p_1[1..8] CONC b"1000"),
        mem_sel),

    sparc_add_2_uv = MUX_4(STRING[12]bit)(
        (b"00000000000001"),
        (b"0000" CONC x_p_1[1..6] CONC b"10"),
        (b"00" CONC x3_p_1[1..7] CONC b"100"),
        (b"0" CONC x7_p_1[1..7] CONC b"1000"),
        mem_sel),

    sparc_add_2 = MUX_2(STRING[12]bit)( sparc_add_2_y, sparc_add_2_uv, CASE channel

```

OF y:left
ELSE right
ESAC).

sparc_oct_add_factor = (sparc_add_1[S_SPA(b'00000000" CONC sparc_add_2)](2)).

#signals when write must start delayed 1 tu for use in zero_hh#

addr_col_read_flag = CASE addr_col_read[2]#decode to bool#
OF count_carry:1
ELSE 1
ESAC,

write_latency = CASE (addr_row_read[1], addr_col_read[1])
OF (row/2,col/(conv2d_latency-1)):1
ELSE 1
ESAC,

read_done = CASE (addr_row_read[2], addr_col_read_flag) #read input data done#
OF (count_carry,1):1
ELSE 1
ESAC,

zero_hh = CAST(!_load)/(NOT zero_hh_bool),

read_valid = CAST(!_load)/(NOT read_done_bool),

start_write_col = DFF_NO_LOAD(!_load)[ck,reset,zero_hh,read], #1 tu after zero_hh#

```

read_mux = CASE (y_done,uv_done,oclave_finished,channel)
OF
  (t.f,write,y)((t.f,write,u):tres, #base_u#
  ((t.f,write,u)((t.f,write,v):qualro, #base_v#
  ((t.f,write,y):dos #base_y#
ELSE uno
ESAC,

```

```

write_mux = CASE zero_hh
OF write:uno,
  read:
    CASE channel
    OF y:dos, #base_y#
      u:tres, #base_u#
      v:qualro #base_v#
    ESAC
  ESAC

```

JOIN

#note that all the counters have to be reset at the end of an octave, ie on octave_finished#
 (ck,oclave_reset,oclave_col_length) ->addr_col_read, #the row&col counts for the read address#
 (ck,oclave_reset,oclave_row_length,addr_col_read[2]) ->addr_row_read,

```

(ck,oclave_reset,write_latency,t)->zero_hh_bool,

```

```

(ck,oclave_reset,read_done,t) ->read_done_bool,

```

#w&r addresses for sparc mem#

```

(ck,reset,PDF1[bool,conv2d_latency-1])(ck,reset,addr_col_read_flag.f).write_mux,sparc_oct_add_factor,base_u,base_v)
->write_addr,

```


- 472 -

```
(ck,reset,addr_col_read_flag,read_mux,spare_oct_add_factor,base_u,base_v) ->read_addr,
```

```
(ck,reset,direction,channel,octave,write_addr,read_addr,zero_hh)->mem_control.
```

```
OUTPUT( mem_control,zero_hh, read_valid,addr_row_read[2],addr_col_read)
END.
```

```
#the basic 2d convolver for transform, rows first then cols.#
```

```
FN CONV_2D = (bool:ck,t_reset:reset,t_input:in,t_direction:direction,[4]t_scratch:pdel,
```

```
t_reset:conv_reset,t_count_control:row_lag,(t_col,t_count_control):addr_col_read)
```

```
->
```

```
(t_input,t_memport,t_count_control,t_count_control,t_count_control):
```

```
#forward direction outputs in row form #
# HH HG HH HG .... #
# HG GG HG GG .... #
# HH HG HH HG .... #
# HG GG HG GG .... #
```

```
#the inverse convolver returns the raster scan format output data#
```

```
#the convolver automatically returns a 3 octave transform#
```

```
BEGIN
```

```
FN CH_PORT = ([4]t_scratch,t_col),t_col)
```

```
->
```

```
t_memport:REFORM.
```

```
MAKE CONV_ROW:conv_row,
CONV_COL:conv_col.
```

```
LET
```

```

row_reset = CASE direction
OF forward:conv_reset,
   inverse:PDF1(l_reset,1)(ck,no_reset,conv_reset,rs1) #pipeline delays in col_conv#
ESAC,
col_reset = CASE direction
OF forward:PDF1(l_reset,3)(ck,no_reset,conv_reset,rs1),
   inverse:conv_reset #pipeline delays in row_conv#
ESAC,

col_flag = DFM(l_count_control)(ck,addr_col_read[2],PDF1(l_count_control,1)(ck,reset,addr_col_read[2],
count_0), CAST[bool](direction),

row_control = DFM(l_count_control)(ck,PDF1(l_count_control,3)(ck,reset,row_flag,count_0),
row_flag, CAST[bool](direction),

direction_sel = CASE direction #mux control for the in/out data mux's#
OF forward:left,
   inverse:right
ESAC,

col_count = MUX_2((l_col,l_count_control))(
PDF1((l_col,l_count_control),3)(ck,reset,addr_col_read,(col0,count_rsl)),
addr_col_read,
direction_sel),

#pipeline delays for the convolver values and input value#
del_conv_col=DFF_NO_LOAD(l_input)(ck,reset,conv_col[1],input0),

del_conv_row=DFF_NO_LOAD(l_input)(ck,reset,conv_row,input0),

del_in = DFF_NO_LOAD(l_input)(ck,reset,in,input0).

```

JOIN

(ck,row_reset,direction,MUX_2(t_input)(del_in,del_conv_col,direction_sel),col_flag) ->conv_row,

(ck,col_reset,direction,MUX_2(t_input)(del_conv_row,del_in,direction_sel),pdel,row_control,col_count) ->conv_col.

OUTPUT (MUX_2(t_input)(del_conv_col,del_conv_row,direction_sel),CH_PORT(conv_col[2],col_count[1]),row_control,col_count[2],col_flag)
END.

1d col convolver, with control

FN CONV_COL = (bool:ck,t_reset:reset,t_direction:direction,t_input:in,
{4}t_scratch:pdel,t_count_control:row_flag,
(t_col,t_count_control):col_count)

->

(t_input,{4}t_scratch,t_col):

#input is data in and, pdel, out from line-delay memories#

out is (G,H), and line delay out port. The row counter is started 1 cycle later to allow for#

#pipeline delay between MULTIPLIER and this unit #

BEGIN

a %2 line by line resetable counter for the state machines, out->one on rst#

#carry active on last element of row#

MAC COUNT_2 = (bool:ck,t_reset:reset,t_count_control:carry)

->

t_count_2:

BEGIN

- 475 -

```

MAKE DFF_NO_LOAD[l_count_2]:countdel.
LET countout= CASE (countdel,carry)
  OF  (one,count_carry):two,
      (two,count_carry):one
  ELSE countdel
  ESAC.
JOIN (ck,reset,countout,one) ->countdel.
OUTPUT countdel
END.

```

```

#the code for the convolver#
MAKE MULT_ADD:mult_add,
  [4]DFF1[l_scratch]:pdel_in,
  [4]DFF1[l_scratch]:pdel_out,
  COUNT_2:count.

```

```

# now the state machines to control the convolver#
#First the and gates#

```

```

LET      reset_row=DFF1[l_reset](ck,reset).      #starts row counter 1 cycle after frame start#
#we want the row counter to be 1 cycle behind the col counter for the delay for the#
#pipelined line delay memory#

```

```

col_carry =DFF_NO_LOAD[l_count_control](ck,reset,col_count[2],count_rst).

```

```

#these need to be synchronised to keep the row counter aligned with the data stream#
#also the delay on col_count deglitches the col carryout#

```

```

row_control=row_flag.      #signal for row=0,1,2,3, last row, etc#

```

- 476 -

```

andsel= (CASE direction
  OF   forward: CASE count
        OF   one: pass,
            two: zero
            ESAC,
        inverse: CASE count
        OF   one: zero,
            two: pass
            ESAC
        ESAC,
    CASE row_control
    OF count_0zero
    ELSE pass
    ESAC,
    CASE direction
    OF   forward: CASE row_control
        OF count_0zero
        ELSE pass
        ESAC,
        inverse: pass
        ESAC),
#now the add/sub control for the convolver address#
addsel= CASE count
  OF one: (add, add, add, sub),
  two: (add, sub, add, add)
  ESAC,

```

```

#now the mux control#
centermuxsel=

CASE direction
OF forward: CASE count
    OF one:(left,right),
       two:(right,left)
    ESAC,
    Inverse: CASE count
    OF one:(right,left),
       two:(left,right)
    ESAC

ESAC,

#the perfect reconstruction output#
#the addmuxsel signal#
muxandsel =

CASE direction
OF forward: (andsel[2],pass,andse[2]),
    Inverse: (pass,andse[2], CASE row_control
    OF count_1 zero
    ELSE pass
    ESAC)

ESAC,

muxsel= CASE direction
OF forward: (uno,

CASE row_control
OF count_0: dos,
    count_carry: tres
    ELSE uno
    ESAC,

CASE row_control
OF count_0: tres,

```

- 478 -

```

count_carry:quatro
ELSE dos
ESAC).

```

```

inverse: ( CASE row_control
OF count_0: dos,
count_1: quatro,
count_carry: dos,
count_lm1: tres
ELSE dos
ESAC,

```

```

CASE row_control
OF count_0: tres,
count_carry: dos
ELSE uno
ESAC,

```

```

uno)

```

```

ESAC.

```

```

LET

```

```

#ACTEL#

```

```

wr_addr = DF1[1: col](ck, DF1[1: col](ck, col_count[1])).
#need 2 delays between wr and rd addr#
rd_addr = col_count[1].
#address for 1 line delay memory#

```

```

#join the control signals to the mult_add block#
JOIN (ck, reset_row, col_carry) -> count1,

```

```

(ck, reset, in, andsel, centermuxsel, muxsel, muxandsel, addsel, direction, pdel_out) -> mult_add.

```

```

FOR INT k=1..4 JOIN
    (ck_mult_addr[k])->pdcl_in[k],    #delay to catch the write address#
    (ck_pdel[k])    ->pdcl_out[k].    #read delay to match MULT delay#

#ACTEL HACK#
LET gh_select = CASE (direction,DF1(t_count_2)(ck_count))
    OF      (inverse,one)((forward,two):right,
            (inverse,two)((forward,one):left
            ESAC,

    gh_out = MUX_2(t_scratch)(pdcl_in[4],DF1(t_scratch)(ck_pdel_out[1]),gh_select),
    shift_const= CASE direction
    OF inverse: CASE DF1(t_count_control)(ck_row_control)
        OF (count_1|count_2):shift3
        ELSE shift4
        ESAC,
        forward: shift5
        ESAC.
    OUTPUT (ROUND_BITS(gh_out,shift_const),(pdcl_in,wr_addr#rd_addr#))    #LOCAM11#
    END.
    #the 1d convolver, with control and coeff extend#

FN CONV_ROW =(bool:ck,t_reset:reset,t_direction:direction,t_input,in,t_count_control:col_flag)
->
    t_input:
    # out is (G,H). The row counter is started 1 cycle later to allow for#
    #pipeline delay between MULTIPLIER and this unit #
    #the strings give the col & row lengths for this octave#

```



```

BEGIN
    # a %2 line by line resetable counter for the state machines, out->one on rat#
    MAC COUNT_2 = (bool:ck,t_reset:reset)
    ->
        t_count_2:
    BEGIN
        MAKE DFF_NO_LOAD(t_count_2):countdel.
        LET
            countout= CASE (countdel)
                OF (one):two,
                   (two):one
                ESAC.
        JOIN (ck,reset,countout,one) ->countdel.
        OUTPUT countdel
        END.

    #the code for the convolver#
    MAKE MULT_ADD:mult_add,
        [4]DFF(t_scratch):pdel,
        COUNT_2:count.

    # now the state machines to control the convolver#
    #First the and gates#

    LET

        reset_col=DFF(t_reset)(ck,reset),      #starts row counter 1 cycle after frame start#
                                                #makes up for the pipeline delay in MULT#
    #HORIZONTAL DEPENDENT!!#
        col_control=col_flag,                  #flag when col_count=0,1,2,col_length,etc#

```

```

andsel= (CASE direction
OF forward: CASE count
OF one:pass,
two:zero
ESAC,
Inverse: CASE count
OF one:zero,
two:pass
ESAC,
ESAC,
CASE col_control
OF count_0:zero
ELSE pass
ESAC,
CASE direction
OF forward: CASE col_control
OF count_0:zero
ELSE pass
ESAC,
Inverse: pass
ESAC),
#now the add/sub control for the convolver address#
addsel= CASE count
OF one:(add,add,add,sub),
two: (add,sub,add,add)
ESAC,
#now the mux control#

```

```

centermuxsel=      CASE direction
                    OF forward: CASE count
                        OF one:(left,right),
                           two:(right,left)
                            ESAC,
                        inverse: CASE count
                            OF one:(right,left),
                               two:(left,right)
                                ESAC
                                ESAC.

#the addmuxsel signal#
muxandsel =      CASE direction
                    OF forward:(andsel[2],pass,andsel[2]),
                       inverse:(pass,andsel[2], CASE col_control
                                                OF count_1:zero
                                                  ELSE pass
                                                    ESAC)
                                ESAC,
                    CASE direction
                    OF forward:(uno,

CASE col_control
OF count_0:dos,
   count_carry:tres
ELSE uno
ESAC,

CASE col_control
OF count_0:tres,
   count_carry:qualro
ELSE dos

```

ESAC),

```
inverse: (CASE col_control
  OF count_0: dos,
    count_1: quatro,
    count_1m1: tres
  ELSE dos
  ESAC,
```

```
  CASE col_control
  OF count_0: tres,
    count_carry: dos
  ELSE uno
  ESAC,
```

uno)

ESAC.

#join the control signals to the mult_add block#

```
JOIN (ck,reset_col) ->count,
```

#set up the col counters #

```
(ck,reset,in,андаel,centermuxsel,muxsel,muxandsel,addsel,direction,pdel)->mult_add.
```

```
FOR INT j=1..4 JOIN
```

```
(ck,mult_add[j]) ->pdel[j].
```

#pipeline delay for mult-add unit#

#ACTEL HACK#

```
LET gh_select=CASE direction
```

```
  OF inverse: CASE count
    OF one: left,
```

```

        two: right
        ESAC,
        forward: CASE count
        OF one:right,
        two:left
        ESAC
        ESAC,

        gh_out = MUX_2(l_scratch){pdel[4],DF1(l_scratch)(ck, pdel[1]),gh_select},

        rb_select= CASE direction
        OF inverse:CASE col_control
        OF (count_2 | count_3):shift3
        ELSE shift4
        ESAC,
        forward: shift5
        ESAC,

        OUTPUT ROUND_BITS(gh_out,rb_select)
        END.

        #some sling macros#
        MAC EQ_US = (STRING[INT n]bit: a b)
        ->
        bool: BIOPEQ_US.

        #ACTEL 8 bit comparator macro#
        FN ICMP8 = (STRING[8]bit: a b)
        ->
        bool: EQ_US[8](a,b).

```

```

#.....#
#A set of boolean ,ie gate level counters
#.....#

#.....#
#The basic toggle flip-flop plus and gate for a synchronous counter #
#input t is the toggle ,outputs are q and tc (toggle for next counter#
#stage
#.....#

MAC_BASIC_COUNT = (bool:ck ,!_reset:reset,bool: tog)
->
[2]bool:

BEGIN

MAKE DFF_NO_LOAD[bool]:dlat,
XOR :xor,
AND :and.

JOIN (ck,reset,xor,!)->dlat,
(dlat,tog) ->and,
(tog,dlat) ->xor.
OUTPUT (dlat,and)

END.

#.....#
# The n-bit macro counter generator, en is the enable, the outputs #
#are msb(bit 1).....lsb,carry. This is the same order as ELLA strings are stored#
#.....#

MAC_COUNT_SYNC(INT n) = (bool:ck,!_reset:reset,bool: en )

```

```

->
(LET out = BASIC_COUNT(ck,reset,en) .
    (n)bool,bool);

    OUTPUT IF n=1
    THEN (i)out[1],out[2])
    ELSE (LET outn = COUNT_SYNC[n-1])(ck,reset,out[2]) .
        OUTPUT (outn[1] CONC out[1],outn[2])
    )
    FI
).

#a mod 2^xsize counter#
MAC MOD2_COUNTER_COL = (bool:ck,t_reset:reset)
->

BEGIN
    MAC S_TO_C = (STRING[xsize]bit:n)
    (t_col):

    MAKE COUNT_SYNC[xsize]:count,
    BOOL_STRING[xsize]:b_s.
    (flag,t_col):BIOP TRANSFORM_US.
->

JOIN (ck,reset,i) ->count, #count always enabled#
    count[i]->b_s.
    OUTPUT (S_TO_C b_s)[2]
    END.

#a mod 2^ysize counter#
MAC MOD2_COUNTER_ROW = (bool:ck,t_reset:reset,bool:en)

```

- 487 -

```

->
(l_row):

MAC S_TO_R = (STRING[ysize]bit.in)

->
((flag,l_row):BIOP TRANSFORM_US.

MAKE COUNT_SYNC[ysize].count,
  BOOL_STRING[ysize] b_s.

JOIN (ck,reset,en) ->count,
  count[1] ->b_s.
OUTPUT (S_TO_R b_s)[2]
END.

#the basic mod col_length counter, to be synthesized#
MAC BASE_COUNTER_COL = (bool:ck,l_reset:reset,STRING[ysize]bit:octave_cnt_length)
->
(l_col,l_count_control):

BEGIN

MAC C_TO_S = (l_col: in)

->
((flag,STRING[ysize]bit): BIOP TRANSFORM_US.

MAC FINAL_COUNT = (l_col:in,STRING[ysize]bit:octave_cnt_length)
->
  l_count_control:

BEGIN
LET in_us = (C_TO_S ln)[2],
  lsb=in_us[ysize].
#OUTPUT CASE EQ_US(in_us[1..ysize-1],octave_cnt_length[1..ysize-1]) the msb's are the same#
#ACTEL#

```



```

OUTPUT CASE ICMP8(in_us[1..xsize-1],octave_cnt_length[1..xsize-1]) #the msb's are the same#
OF t: CASE lsb
    #so check the lsb#
    OF b'1':count_carry, #count odd, so must be length#
        b'0':count_lm1 #count is even so must be length-1#
    ESAC
    ELSE count_rst
    ESAC
END.

MAKE MOD2_COUNTER_COL:mod2_count,
FINAL_COUNT:final_count.

JOIN (mod2_count,octave_cnt_length) ->final_count,
      (ck,CASE reset #system reset or delayed carryout reset#
      OF rst: rst
      ELSECASE OFF_NO_LOAD(l_count_control)(ck,reset,final_count,count_0) #latch to avoid glitches#
      OF count_carry:rst
      ELSE no_rst
      ESAC
      ESAC) ->mod2_count.

OUTPUT (mod2_count,final_count)
END.

FN COL_COUNT_ST = (bool:ck,l_reset:reset,STRING[size]bit:octave_cnt_length)
-->
      (l_col,l_count_control):
      #count value , and flag for count=0,1,2,col_length-1,col_length#

BEGIN
    MAKE BASE_COUNTER_COL:base_col.

    LET count_control = CASE reset

```

```

OF rst:count_0
  ELSE CASE base_col{1}
    OF
      col0:count_0,
      col1:count_1,
      col2:count_2,
      col3:count_3
    ELSE base_col{2}
      ESAC

```

```

      ESAC.

```

```

JOIN (ck, reset, octave_cnt_length) -> base_col.

```

```

OUTPUT (base_col{1}, count_control)

```

```

END.

```

#the basic mod row_length counter, to be synthesised#

```

MAC BASE_COUNTER_ROW = (bool:ck{1}, reset:reset, bool:en, STRING[ysize]bit:octave_cnt_length, l_count_control:col_carry)
->

```

```

(l_row, l_count_control):

```

```

BEGIN

```

```

MAC R_TO_S = (l_row:ln)

```

```

->

```

```

(flag, STRING[ysize]bit): BIOP TRANSFORM_US.

```

```

MAC FINAL_COUNT = (l_row:ln, STRING[ysize]bit:octave_cnt_length)

```

```

->

```

```

l_count_control:

```

```

BEGIN

```

```

LET in_us = (R_TO_S ln){2}.

```

```

  lsb=ln_us[ysize].

```

```

#OUTPUT CASE EQ_US(in_us[1..ysize-1], octave_cnt_length[1..ysize-1]) the msb's are the same#

```

```

#ACTEL#
OUTPUT CASE ICMP8(in_us[1..ysize-1],octave_cnt_length[1..ysize-1]) #the msb's are the same#
OF 1: CASE lsb
    OF b'1:count_carry, #count odd, so must be length#
        b'0:count_lm1 #count is even so must be length-1#
            ESAC
        ELSE count_rst
            ESAC
    END.
MAKE MOD2_COUNTER_ROW:mod2_count,
FINAL_COUNT:final_count.

#need to delay the reset at end of count signal till end of final row#
#WAS DFF WITH reset#
LET count_reset = DF1(!_reset)(ck,CASE(final_count,col_carry) #last row/last col#
    OF (count_carry,count_carry):rst #latch to avoid glitches#
    ELSE no_rst
        ESAC).

JOIN (mod2_count,octave_cnt_length) -> final_count,
(ck,CASE reset
    OF rst: rst
        ELSE count_reset
            ESAC,en) -> mod2_count.
OUTPUT (mod2_count,final_count)
END.

FN ROW_COUNT_CARRY_ST = (bool:ck,!_reset:reset,STRING|size|bit|octave_cnt_length,!_count_control:col_carry)
->
(!_row,!_count_control):

```

```

BEGIN
    MAKE BASE_COUNTER_ROW_base_row.
    LET count_control = CASE reset
        OF rst:count_0
            ELSE CASE base_row[1]
                OF row/0:count_0,
                    row/1:count_1,
                    row/2:count_2,
                    row/3:count_3
                ELSE base_row[2]
                ESAC
            ESAC.

```

```

JOIN (ck,reset,CASE col_carry
    OF count_carry:1
    ELSE:1
    ESAC,octave_cnt_length,col_carry) -->base_row.
OUTPUT (base_row[1],count_control)
END.

```

#the discrete wavelet transform chip/ multi-octave/2d transform with edge compensation#
 #when ext & csl are both low latch the setup params from the nbus(active low), as follows#

```

#ad[1..4]    select function#
# 0000 load max_octaves, luminance/colour, forward/inversebar#
# 0001 load yimage#
# 0010 load ximage#
#jump table values#
# 0011 load ximage+1#
# 0100 load 3ximage+3#
# 0101 load 7ximage+7#

```

```

#      0110  load base u addr#
#      0111  load base v addr#

#adl[21..22]  max_octaves#
#adl[23]luminance/crominancebar active low, 1 is luminance, 0 is colour#
#adl[24]forward/inversebar active low, 1 is forward, 0 is inverse#

#adl[5..24]  data (bit 24 lsb)#

FN ST_OCT = (STRING[2]bit:st)
->          (llag,l_octave): BIOP TRANSFORM_US.

FN OCT_ST = (l_octave:sl)
->          (llag,STRING[2]bit): BIOP TRANSFORM_US.

FN DWT = (bool:ck_in,l_reset:reset_in,l_input:in_in,bool:extwritel_in csl_in, STRING[24]bit:adl,
          l_input:sparc_mem_in,[4]l_scratch:pdcl_in)

->          (l_input#out IDWT data#[3]l_load#valid out IDWT data,y,u,v#,
          [3]l_load#valid in DWT data y,u,v#,
          l_sparcport#sparc_data_addr,etc#,
          l_memport#pdcl_data_out#):

BEGIN
MAKE CONV_2D:conv_2d,
ADDR_GEN_NOSCRATCH:addr_gen,
#active low clock &enable latches#

```

```

[2]DLE1D:max_octave_sl,
DLE1D:channel_factor_sl,
DLE1D:dir,
[9]DLE1D:col_length_s,
[9]DLE1D:row_length_s,
[9]DLE1D:x_p_1,
[11]DLE1D:x3_p_1,
[12]DLE1D:x7_p_1,
[19]DLE1D:base_u,
[19]DLE1D:base_v,
#active low 3X8 decoder#
DEC3X8A                                :decodel,
#the octave control#
DFF_INIT(!_octave):oclave ,
DFF_INIT(!_channel):channel ,
JKFF:row_carry_ff,
#pads#
INBUF[STRING(24bit):ad1_out,
CLKBUF:ck,
INBUF[bool]:extwritel_csl,
INBUF(!_reset):reset,
INBUF(!_input):in_sparc_mem,
INBUF[4]:_scratch]:pdel,
OBHS(!_input):out1,
OBHS[3]:_load]:out2 out3,
OBHS(!_sparcport):out4,
OBHS(!_memport):out5.
#must delay the write control to match the data output of conv_2d, ie by conv2d_latency#
LET
#set up the control params#

```

```
max_ocl = (ST_OCT_BOOL_STRING[2]max_octave_sl)[2],
```

```
channel_factor = CAST[(l_channel_factor)channel_factor_sl,
```

```
col_length = BOOL_STRING[9] col_length_sl,
```

```
row_length = BOOL_STRING[9] row_length_sl,
```

```
direction = CASE dir
```

```
OF f:forward,
```

```
t:inverse
```

```
ESAC,
```

```
#set up the octave params#
```

```
convcol_row = conv_2d[3],
```

```
convcol_col = conv_2d[4],
```

```
convrow_col = conv_2d[5],
```

```
#signals that conv_col, for forward, or conv_row, for inverse, has finished that octave#  
#and selects the next octave value and the sub-image sizes#
```

```
octave_finished = CASE direction
```

```
OF forward:CASE (row_carry_fl,convcol_row,convcol_col)
```

```
OF (t.count_2,count_2):write #row then col, gives write latency#
```

```
ELSE read
```

```
ESAC,
```

```
inverse:CASE (row_carry_fl,convcol_row,convrow_col)
```

```
OF (t.count_2,count_3):write #extra row as col then row#
```

```
ELSE read
```

```
ESAC
```

```
ESAC,
```

```
#max octaves for ulv#
```

- 495 -

```

max_oct_1 = CASE max_oct
  OF oct/1:oct/0,
     oct/2:oct/1,
     oct/3:oct/2
  ESAC,

y_done = CASE (channel,(OCT_ST octave)[2] EQ_US CASE direction
  OF forward:CAST[STRING (2*1)max_octave_st,
     inverse:b'00"
  ESAC)

  OF (y.1)1
  ELSE 1
  ESAC,

uv_done = CASE (channel,(OCT_ST octave)[2] EQ_US CASE direction
  OF forward:(OCT_ST max_oct_1)[2],
     inverse:b'00"
  ESAC)

  OF (u/v.1)1
  ELSE 1
  ESAC,

next= (SEQ
  VAR new_oct:=octave,
  new_channel:=channel;
  CASE direction
  OF forward:(CASE octave
    OF oct/0:new_oct:=oct/1,
       oct/1:new_oct:=oct/2,
       oct/2:new_oct:=oct/3

```


ESAC;

```

CASE (y_done,uv_done)
OF (1,bool)((bool.1):new_oct:=oct/0
ELSE
ESAC
).

```

inverse:(CASE octave

```

OF oct/3:new_oct:=oct/2,
oct/2:new_oct:=oct/1,
oct/1:new_oct:=oct/0
ESAC;

```

CASE channel

```

OF y: CASE octave

```

```

OF oct/0: CASE channel_factor #watch for colour#

```

```

OF luminance:new_oct:=max_oct
ELSE
new_oct:=max_oct_1
ESAC

```

ELSE

ESAC,

u: CASE octave

```

OF oct/0:new_oct:=max_oct_1

```

ELSE

ESAC,

v: CASE octave

```

OF oct/0:new_oct:=max_oct #move to y#

```

ELSE

ESAC

ESAC)

```

ESAC;
CASE channel_factor
OF luminance:new_channel:=y,
   color: (CASE (channel,y_done)
OF (y,1):new_channel:=u
ELSE
ESAC;
CASE (channel,uv_done)
OF (u,1):new_channel:=v,
   (v,1):new_channel:=y
ELSE
ESAC)
ELSE
ESAC;
OUTPUT (new_oct,new_channel)
),

octave_sel = CASE (octave,channel) #the block size divides by 2 every octave#
OF (oct/0,y):uno, #the uv image starts 1/4 size#
   (oct/1,y):(oct/0,u|v):dos,
   (oct/2,y):(oct/1,u|v):tres,
   (oct/3,y):(oct/2,u|v):quatro
ESAC,
octave_row_length = MUX_4(STRING [ysize|bit](row_length,b'0' CONC row_length[1..ysize-1],
   b'00' CONC row_length[1..ysize-2],
   b'000' CONC row_length[1..ysize-3],octave_sel),
octave_col_length = MUX_4(STRING [xsize|bit](col_length,b'0' CONC col_length[1..xsize-1],
   b'00' CONC col_length[1..xsize-2],
   b'000' CONC col_length[1..xsize-3],octave_sel),

```

```

#load next octave, either on system reset, or write finished#
load_octave= CASE reset
              OF rst:write
              ELSE octave_finished
              ESAC,
#reset the convolvers at the end of an octave, ready for the next octave#
#latch pulse to clean it, note 2 reset pulses at frame start#
#cant glitch as reset&octave_finished dont change at similar times#
conv_reset = CASE reset
              OF rst:rst
              ELSE CASE DIFF_NO_LOAD[!_load](ck.reset, octave_finished.read)
              OF write:rst
              ELSE no_rst
              ESAC
              ESAC,

#latch control data off nubus, latch control is active low#
gl = CASE (extwrite!.cs)
    OF (!,0):1
    ELSE 1
    ESAC,

sparc_w=addr_gen[1][2][1]. #write addresses#
input_mux=addr_gen[1][1]. #input_mux#
sparc_r=addr_gen[1][2][2]. #read addresses#
sparc_rw = addr_gen[1][2][3].

```

```

inverse_out = CASE (direction,octave)
OF (inverse_oct/0):CASE (channel,addr_gen[2])
  OF (y,write):(write,read,read),
     (u,write):(read,write,read),
     (v,write):(read,read,write)
  ELSE (read,read,read)
  ESAC,
  (forward_oct/0):(read,read,read)
ELSE (read,read,read)
ESAC,
forward_in = CASE direction
OF forward:CASE (channel,octave,addr_gen[3])
  OF (y,oct/0,read):(read,write,write),
     (u,oct/0,read):(write,read,write),
     (v,oct/0,read):(write,write,read)
  ELSE (write,write,write)
  ESAC,
  Inverse:(write,write,write)
ESAC.

JOIN
#in pads#
  ck_in      ->ck,
  reset_in->reset,
  extwrite1_in ->extwrite1,
  cal_in      ->cal,
  adl         ->adl_out,
  in_in       ->in,
  spare_mem_in ->spare_mem,
  pdel_in     ->pdel,
#out pads#

```

```

conv_2d[1]    ->out1,
inverse_out   ->out2,
forward_in    ->out3,
addr_gen[1][2] ->out4,
conv_2d[2] ->out5,

```

#the control section#

```

(CAST[bool]ad[4],CAST[bool]ad[3],CAST[bool]ad[2]) ->decode1, #active low out5#

(g1,decode1[1],BIT_BOOLadl_out[21]) ->max_octave_sl[1],
(g1,decode1[1],BIT_BOOLadl_out[22]) ->max_octave_sl[2],

(g1,decode1[1],BIT_BOOLadl_out[23]) ->channel_factor_sl,
(g1,decode1[1],BIT_BOOLadl_out[24]) ->dlr.

FOR INT j=1..9 JOIN
  (g1,decode1[2],BIT_BOOLadl_out[15+j]) ->col_length_sl[j],
  (g1,decode1[3],BIT_BOOLadl_out[15+j]) ->row_length_sl[j],
  (g1,decode1[4],BIT_BOOLadl_out[15+j]) ->x_p_1[j].

FOR INT j=1..11 JOIN
  (g1,decode1[5],BIT_BOOLadl_out[13+j]) ->x3_p_1[j].

FOR INT j=1..12 JOIN
  (g1,decode1[6],BIT_BOOLadl_out[12+j]) ->x7_p_1[j].

FOR INT j=1..19 JOIN
  (g1,decode1[7],BIT_BOOLadl_out[5+j]) ->base_u[j],
  (g1,decode1[8],BIT_BOOLadl_out[5+j]) ->base_v[j].

```

#sets a flag when row counter moves onto next frame#

```

JOIN
  (ck,conv_reset,CASE convcol_row
    OF count_carry:1
    ELSE 1

```

```
ESAC,!)
->row_carry_H,
```

#load the new octave,after the current octave has finished writing#
on initial reset must load with starting octave value which depends on direction and channel#

```

(ck,no_rst,load_octave,CASE reset
  OF no_rst,next[1]
  ELSE CASE (direction,channel) #initial octave#
    OF (forward,t_channel):oct/0,
       (inverse,y):max_oct,
       (inverse,u,v):max_oct_1
    ESAC
    ESAC,oct/0)
  ->octave, #next octave#
(ck,no_rst,load_octave,CASE reset
  OF no_rst,next[2]
  ELSE y
  ESAC,y)
  ->channel, #next channel#

```

```

        (ck,reset,MUX_2(i_input))(in,sparc_mem,CASE input_mux $input_mux#
        OF dwt_in:left,
            sparc_in:right
            ESAC)
        ,direction,pdel,conv_reset,addr_gen(4),addr_gen(5))
        -->conv_2d,

```

```
(ck,reset,direction,channel,BOOL_STRING[9]x_p_1,BOOL_STRING[11]x3_p_1,BOOL_STRING[12]x7_p_1,octave_row_length,
octave_col_length,conv_reset,octave_y_done,uv_done,octave_finished,BOOL_STRING[19]base_u,BOOL_STRING[19]base_v)
->addr_gen.
```

OUTPUT: (out1 ,out2,out3,out4,out5)

END.

```
EN DWT TEST = (bool:ck in, t, resel:resel in, t, input:in in, bool:extwritel in, csl in, t, sparc_addr:reg_sel value)
END.
```

```

->
        (t_input[3]t_load[3]t_load):

BEGIN
    FN SPARC_MEM = (t_input.in,t_sparc_addr.wr_addr,t_sparc_addr.rd_addr,t_load.rw_sparc#.t_cs.cs#)
->
        t_input:
        RAM(input/0).

    MAKE DWT:dwt,
        SPARC_MEM:sparc_mem,
        LINE_DELAY(t_scratch):line_delay.

    LET  data_out=dwt[1],
        sparc_port=dwt[4],
        line_delay_port = dwt[5].

    JOIN
        (ck_in,reset_in,in_in,extwritel_in,cal_in,(SPA_S reg_sel)[2][16..19]CONC b"1" CONC(NOT_B (SPA_S value)[2]), sparc_mem,line_delay)
        ->dwt,
        (data_out,sparc_port[1],sparc_port[2],sparc_port[3]#,sparc_port[4]#)    ->sparc_mem,
        (line_delay_port[1],line_delay_port[2],line_delay_port[3],write)    ->line_delay.

    OUTPUT  dwt[1..3]
    END.

# some basic macros for the convolver, assume these will#
#be synthesised into leaf cells#
#the actel MX4 mux cell#
FN NOT = (bool:in)
->
    bool:CASE in OF 1:1,1:1ESAC.

```

MAC MX_4[TYPE ty]=(ty:in1 in2 in3 in4, [2]bool:sel)

->

ty:

CASE sel
OF (t1).in1,
(t1).in2,
(t1).in3,
(t1).in4

ESAC.

#the actel GMX4 mux cell#

MAC GMX4[TYPE ty]=(ty:in1 in2 in3 in4, [2]bool:sel)

->

ty:

CASE sel
OF (t1).in1,
(t1).in2,
(t1).in3,
(t1).in4

ESAC.

MAC MXT[TYPE ty]=(ty:a b c d, bool:soa sob s1)

->

ty:

CASE s1
OF t:CASE soa
OF t1b
ELSE a
ESAC,

t:CASE sob


```

OF l:d
ELSE c
ESAC

```

```

ESAC.

```

```

MAC ENCODE4_2 = (l_mux4:in)

```

```

->

```

```

[2]bool:

```

```

CASE in

```

```

OF uno:(l,f),

```

```

dos:(f,f),

```

```

tres:(f,f),

```

```

quatro:(f,f)

```

```

ESAC.

```

```

MAC ENCODE3_2 = (l_mux3:in)

```

```

->

```

```

[2]bool:

```

```

CASE in

```

```

OF l:(f,f),

```

```

c:(f,f),

```

```

r:(f,f)

```

```

ESAC.

```

```

FN DEC3X8A = (bool:a b c)

```

```

->

```

```

[8]bool:

```

```

CASE (a,b,c)

```

```

OF (f,f,f):(f,f,f,f,f,f,f,f),

```

```

(f,f,f):(f,f,f,f,f,f,f,f),

```

```

(f,f,f):(f,f,f,f,f,f,f,f),

```

```

(f,f,f):(f,f,f,f,f,f,f,f),

```

- 505 -

```

((1,0):(0,1,1,1,1,1,0),
(0,0):(0,1,1,1,1,1,0),
(1,1):(0,1,1,1,1,1,0),
(1,1):(0,1,1,1,1,1,0)

```

ESAC.

MAC_MUX_2(TYPE 1)=(in1 in2, l_mux:sel)

->

```

t: CASE sel
    OF left:in1,
       right:in2

```

ESAC.

MAC_MUX_3(TYPE 1)=(in1 in2 in3, l_mux3:sel)

->

```

t: MX_4((in1,in2,in3,in1,ENCODE3_2 sel).

```

COM

MAC_MUX_4(TYPE 1)=(in1 in2 in3 in4, l_mux4:sel)

->

```

t: CASE sel
    OF uno:in1,
       dos:in2,
       tres:in3,
       quatro:in4

```

ESAC.

MOC

```
MAC MUX_4(TYPE 1)=(t.in1 in2 in3 in4, t_mux4.sel)
```

```
->
```

```
t:
```

```
MX_4(t)(in1,in2,in3,in4,ENCODE4_2 sel).
```

```
FN AND2 = (bool:a b)
```

```
->
```

```
bool:BIOP AND.
```

```
MAC GNAND2 = (bool:a b)
```

```
->
```

```
bool:NOT AND2(a,b).
```

```
MAC AND_2 = (t_scratch.in, t_and.sel)
```

```
->
```

```
t_scratch:
```

```
BEGIN
```

```
LET in_s = (1 TO S(scratch_exp)in)[2].
```

```
sel_s = CAST[bool]sel.
```

```
OUTPUT (S_TO_I(scratch_exp)BOOL_STRING(scratch_exp) ((INT)-1..scratch_exp)AND2(BIT_BOOL in_s[sel_s]))[2]
```

```
END.
```

```
FN XOR = (bool: a b)
```

```
->
```

```
bool:
```

```
CASE (a,b)
```

```
OF (1,1)|(1,1):1
```

```
ELSE 1
```

```
ESAC.
```

```
MAC XOR_B[INT n] = (STRING[n]bit:a b)
->
STRING[n]bit:BIOP XOR.

MAC NOT_B = (STRING[INT n]bit:a)
->
STRING[n]bit:BIOP NOT.

MAC XNOR_B = (STRING[INT n]bit:a b)
->
STRING[n]bit:
NOT_B XOR_B[n](a,b).

FN AND = (bool: a b)
->
bool:
CASE (a,b)
OF (1,1)1
ELSE 1
ESAC.

MAC DEL[TYPE t] = (t)
->
t:DELAY(?t,1).

#a general dff same as DFF_NO_LOAD#
MAC DFF [TYPE t]=(bool:ck,t_reset:reset,t in init_value)
->
t:
BEGIN
```

```
MAKE DEL(i):del.  
JOIN in->del.  
OUTPUT CASE reset  
  OF rst:init_value  
    ELSE del  
      ESAC  
END.
```

```
#a general dff#  
MAC DF1 (TYPE i)=(bool:ck,i:in)  
->  
i:  
BEGIN  
  MAKE DEL(i):del.  
  JOIN in->del.  
  OUTPUT del  
END.
```

```
#a general latch#  
MAC DL1 (TYPE ty)=(bool:ck,ty:in)  
->  
ty:  
BEGIN  
  MAKE DEL(ty):del.  
  JOIN CASE ck  
    OF t:in  
      ELSE del  
        ESAC ->del.  
  OUTPUT CASE ck  
    OF t:in
```

- 509 -

```
ELSE del
ESAC
```

```
END.
```

```
#a general d latch#
```

```
MAC LATCH (TYPE t)=(bool:ck,l_load:load,t:in)
```

```
->
```

```
t:
```

```
BEGIN
```

```
MAKE DEL(t):del.
```

```
LET out=CASE load
```

```
OF write:in
```

```
ELSE del
```

```
ESAC.
```

```
JOIN out->del.
```

```
OUTPUT out
```

```
END.
```

```
#an ACTEL D LATCH#
```

```
MAC DLE1D = (bool:ckl loadl, bool:in)
```

```
->
```

```
bool:#qnr#
```

```
NOT LATCH(bool)(NOT ckl, CASE loadl
```

```
OF l:write
```

```
ELSE read
```

```
ESAC, in).
```

```
MAC PDF1(TYPE t,INT n) = (bool:ck,l_resel:resel,t:in initial_value)
```

```
->
```

```
t:
```

```
IF n=0 THEN DFF1(ck,resel,in,initial_value)
```

```

ELSE PDF1((n-1))(ck,reset,DFF(i))(ck,reset, in,initial_value),initial_value)
FI.

```

```

# a muxed input dff#
MAC DFM (TYPE ty)=(bool:ck,ty:a b,bool:s)
->
ty:
BEGIN
MAKE DEL(ty):del.
JOIN CASE s
OF t:a,
t:b
ESAC ->del.
OUTPUT del
END.
#a resetable DFF, init value is input parameter#
MAC DFF_INIT(TYPE t)=(bool:ck,t_reset:reset,t_load:load,t_in init_value)
->
t:
BEGIN
MAKE DEL(t):del.
LET out=CASE (load,reset)
OF (write,t_reset):in,
(read,rsf):init_value
ELSE del
ESAC.
JOIN out->del.
OUTPUT CASE reset
OF rst:init_value
ELSE del

```

- 511 -

```

ESAC
END.

```

```

#a resetable JKFF, k input is active low#
FN JKFF=(bool:ck,t_reset:reset,bool:j k)

```

```

->

```

```

bool:

```

```

BEGIN

```

```

MAKE DEL(bool):del.

```

```

LET out=CASE (j,k,reset)

```

```

OF (t,t,no_rst):t,

```

```

(t,t,rst):t,

```

```

(t,t,no_rst):t,

```

```

(t,t,no_rst):del,

```

```

(t,t,no_rst):del,

```

```

(t,t,no_rst):NOT del

```

```

ESAC.

```

```

JOIN out->del.

```

```

OUTPUT CASE reset

```

```

OF rst:/

```

```

ELSE del

```

```

ESAC

```

```

END.

```

```

#a diff resetable non-loadable diff#

```

```

MAC DFF_NO_LOAD(TYPE i)=(bool:ck,t_reset:reset,t,in init_value)

```

```

->

```

```

t:

```

```

BEGIN

```

```

MAKE DEL(i):del.

```

```

JOIN in->del.

```



```

OUTPUT CASE reset
  OF rst_init_value
  ELSE del
  ESAC
END.

```

```

MAC PDEL(TYPE t,INT n) = (t in)

```

```

->
t:

```

```

  IF n=0 THEN DEL(t) in
  ELSE PDEL(t,n-1) DEL(t) in
  FI.

```

#the mem control unit for the DWT chip, outputs the memport values for the sparc, and dwt#
 #inputs datain from these 2 ports and mux's it to the 2d convolver.#

```

MAC MEM_CONTROL_NOSCRATCH = (boolck.t_reset:reset.t_direction:direction.t_channel:channel.t_octave:octave,
  t_sparc_addr:sparc_addr_w_sparc_addr_r.t_loadzero_hh)

```

```

->

```

```

  ((l_input_mux.t_sparcport.t_dwtport#dwt#):

```

```

BEGIN

```

```

#the comb. logic for the control of the i/o ports of the chip#

```

```

LET ports = (SEQ

```

```

  VAR #defaults, so ? doesnt kill previous mem value#

```

```

  rw_sparc:=read,

```

```

  rw_dwt:=read,

```

```

  cs_dwt:=no_select,

```

```

  input_mux:=sparc_in;

```

- 513 -

```

CASE (direction,octave)
  OF
    (forward,oct/0): ( cs_dwt:=select;
                      input_mux:=dwt_in),
    (inverse,oct/0):( CASE zero_hh
                      OF  write {rw_dwt:=write;
                                cs_dwt:=select}
                      ELSE
                        ESAC)

    ESAC;

#rw_sparc=write when ck=1 and zero_hh=write, otherwise = read#
rw_sparc:= CAST[1_load]GNAND2(NOT CAST[bool]zero_hh,ck);

#mux the sparc addr on clock#
sparc_addr = GMX4[1_sparc_addr](sparc_r,sparc_r,sparc_w,sparc_w,ck,0);#

OUTPUT (input_mux, (sparc_addr_w,sparc_addr_r,rw_sparc), #sparc port#
        (rw_dwt,cs_dwt)
        #dwt port#
        )
).
OUTPUT ports
END.

# the basic 1d convolver without the control unit#

MAC MULT_ADD = (bool:ck,1_resel: resel,1_input:in, [3]t_and:andsel, [2]t_mux:centermuxsel,[3]t_mux4:muxsel,
                [3]t_and:muxandsel,[4]t_add:address, t_direction:direction,[4]t_scratch:pdel)

->
[4]t_scratch: #pdel are the outputs from the line delays#

```

```

BEGIN
    MAKE MULTIPLIER:mult,
    [4]ADD_SUB: add.
    #the multiplier outputs#
    LET x3=mult[1],
        x5=mult[2],
        x11=mult[3],
        x19=mult[4],
        x2=mult[5],
        x8=mult[6],
        x30=mult[7],

    #the mux outputs#
    mux1=MUX_4(l_scratch)(x11,x5,x8,x2,muxsel[1]),
    mux2=MUX_4(l_scratch)(x19,x30,x8,scratch0,muxsel[2]),
    mux3=MUX_4(l_scratch)(x11,x5,x8,x2,muxsel[3]),
    centermux=(MUX_2(l_scratch)(pde[1],pde[3],centermuxsel[1]),
    MUX_2(l_scratch)(pde[2],pde[4],centermuxsel[2])),

    # the AND gates zero the adder inputs every 2nd row#
    #the and gate outputs#
    and1=AND_2(pde[2],andsel[1]),
    and2=AND_2(pde[3],andsel[1]),
    and3=AND_2(centermux[1],andsel[2]),
    and4=AND_2(centermux[2],andsel[3]),
    add1in=AND_2(mux1,muxandsel[1]),

```

- 515 -

```

add3in=AND_2(mux3,muxandse[2]).
add4in=AND_2(x3,muxandse[3]).

```

```

JOIN in ->mult.
  (and1,add1in,addse[1]) ->add[1],
  (and3,mux2,addse[2]) ->add[2],
  (and4,add3in,addse[3]) ->add[3],
  (and2,add4in,addse[4]) ->add[4].

```

OUTPUT add

END.

the basic multiplier unit of the convolver

```
MAC MULTIPLIER_ST = (l_input:in)
```

->

```

[7] scratch:
#x3,x5,x11,x19,x2,x8,x30#

```

BEGIN

```
MAC INPUT_TO_S(INT n) = (l_input: in)
```

->

```
(flag,STRING(n*11): BIOP TRANSFORM_S.
```

#the multiplier outputs, fast adder code commented out#

```

LET in_s= (INPUT_TO_S(input_exp)in)[2].
  x2=in_s CONC b'0'.
  x8=in_s CONC b'000'.
  x3 = ADD_S_ACTEL(in_s, x2,b'1),
  x5 = ADD_S_ACTEL(in_s,in_s CONC b'00'b'1 ).
  x11 = ADD_S_ACTEL(x3,x8,b'1).
  x19 = ADD_S_ACTEL(x3,in_s CONC b'0000'b'1).

```

```

x30=ADD_S_ACTEL(x11,x19,b'1).

LET subsignal = (x2,x8, x3,x5, x11,x19,x30).
OUTPUT ((S_TO_[input_exp+2] x3)[2],(S_TO_[input_exp+3] x5)[2],(S_TO_[input_exp+4] x11)[2],
        (S_TO_[input_exp+5] x19)[2],(S_TO_[input_exp+1] x2)[2],(S_TO_[input_exp+3] x8)[2],
        (S_TO_[input_exp+6] x30)[2])
END.
MAC INBUF(TYPE i) = (i:pad)
->
i:#y#pad.

MAC OBHS(TYPE i) = (i:d)
->
i:#pad#d.

FN CLKBUF = (bool:pad)
->
bool:pad.
#MAC SHIFT(INT p) = (STRING[scratch_exp]bit) -> STRING[scratch_exp+p]bit:BIOP SR_S[p].#

MAC ADD_S = (STRING[INT m]bit,STRING[INT n]bit)
->
STRING[IF m>n THEN m+1 ELSE n+1 F]bit:
BIOP PLUS_S.

MAC INV[INT m] = (STRING[m]bit:a)
->
STRING[m]bit:BIOP NOT.

MAC NEG_S = (STRING[INT n]bit)
->

```

```
STRING[n+1]bit:
BIOP NEGATE_S.
```

```
MAC ADD_US = (STRING[INT m]bit,STRING[INT n]bit)
```

```
->
STRING[IF m>n THEN m+1 ELSE n+1 F]bit:
BIOP PLUS_US.
```

```
MAC CARRY = (L_add:in)
```

```
->
STRING[1]bit: CASE in
    OF add:b'0',
       sub:b'1'
    ESAC.
```

```
#acle1 adder macros#
```

```
#an emulation of a fast ACTEL 16 bit adder with active low carries#
FN FADD16 = (STRING[scratch_exp]bit: a b,STRING[1]bit:cinb)
```

```
->
(STRING[scratch_exp]bit,STRING[1]bit):
BEGIN
```

```
LET  a_c = a CONC INV(1)cinb,
      b_c = b CONC INV(1)cinb,
      out = ADD_S(a_c,b_c).
```

```
OUTPUT(out[2..scratch_exp+1],INV(1)B TO S out(1))
END.
```

```
#acle1 1 bit full adder with active low cin and cout#
MAC FA1B = (bit: ain bin cinb)
```

```
->
```

- 518 -

```

(bit,bit):#cob,s#
BEGIN
  LET  a_c = B_TO_S ain CONC INV(1)B_TO_S cinb ,
        b_c = B_TO_S bin CONC INV(1)B_TO_S cinb ,
        out = ADD_US(a_c,b_c).
  OUTPUT(CAST[bit] INV(1) B_TO_S out(1), out(2))
END.

#the actel version of the ADD BIOP's#

MAC ADD_US_ACTEL = (STRING(INT m)bit:ain,STRING(INT n)bit:bin,bit:cinb)
->
  STRING(IF m>=n THEN m+1 ELSE n+1 F)bit:
  BEGIN
    MAKE [IF m>=n THEN m ELSE n F]A1B:sum.

    #unsigned nos so extend by 0#
    LET a_c = IF m>=n THEN ain ELSE ZERO(n-m)b"0" CONC ain F1,
        b_c = IF n>=m THEN bin ELSE ZERO(m-n)b"0" CONC bin F1.
    LET subsignal = sum.

    #lsb#
    JOIN  (a_c[IF m>=n THEN m ELSE n F],b_c[IF m>=n THEN m ELSE n F],cinb) ->sum[IF m>=n THEN m ELSE n F].

    FOR INT j=1..(IF m>=n THEN m ELSE n F) -1
    JOIN (a_c[IF m>=n THEN m ELSE n F] -j,b_c[IF m>=n THEN m ELSE n F] -j, sum[IF m>=n THEN m ELSE n F] -j+1 F1))
    >sum[IF m>=n THEN m ELSE n F] -j.

    OUTPUT  CAST[STRING(IF m>=n THEN m+1 ELSE n+1 F)bit]
    (INV(1) B_TO_S sum[1] F1) CONC

```

```
CAST(STRING(IF m>n THEN m ELSE n F1)bit) (INT j=1..IF m>n THEN m ELSE n F1) sum[j](2))
```

```
END.
```

```
MAC ADD S_ACTEL = (STRING(INT m)bit:ain, STRING(INT n)bit:bin, bit:cinb)
```

```
→
```

```
STRING(IF m>n THEN m+1 ELSE n+1 F1)bit:
```

```
BEGIN
```

```
MAKE (IF m>n THEN m ELSE n F1)FA1Bsum.
```

```
#signed nos eo sign extend #
```

```
LET a_c = IF m>n THEN ain ELSE ALL_SAME(n-m)B_TO_S ain(1) CONC ain F1,
```

```
b_c = IF n>m THEN bin ELSE ALL_SAME(m-n)B_TO_S bin(1) CONC bin F1.
```

```
LET subsignal = sum.
```

```
#lsb#
```

```
JOIN (a_c(IF m>n THEN m ELSE n F1), b_c(IF m>n THEN m ELSE n F1), cinb) →sum(IF m>n THEN m ELSE n F1).
```

```
FOR INT j=1..(IF m>n THEN m ELSE n F1) -1
```

```
JOIN (a_c(IF m>n THEN m ELSE n F1) -j), b_c(IF m>n THEN m ELSE n F1) -j, sum((IF m>n THEN m ELSE n F1) -j+1)(1)) →sum((IF m>n THEN m ELSE n F1) -j).
```

```
OUTPUT CAST(STRING(IF m>n THEN m+1 ELSE n+1 F1)bit)
```

```
((INV(1) B_TO_S sum[1])(1) CONC
```

```
CAST(STRING(IF m>n THEN m ELSE n F1)bit) (INT j=1..IF m>n THEN m ELSE n F1) sum[j](2))
```

```
END.
```

```
FN ROUND_BITS = (l_scratch.in, l_round: select)
```

```
→
```

```
l_input:
```

```
BEGIN
```



```

#THIS ASSUMES THAT THE INPUT_EXP=10!!!!#
#select chooses a round factor of 3, 4, 5#
#the lsb is the right hand of the string.#
#the index 1 of the string is the left hand end, & is the msb#
#so on add ops bit 1 is the carryout#
LET s1= (L_TO_S[scratch_exp]in)[2].

```

```

msb= B_TO_S s1[1].

```

```

selector = CASE select      #case conversion for MUX_3#
OF shift3l,
  shift4x,
  shift5r
ESAC.

```

```

#needs to be a 16 bit output for the adder#

```

```

shift = MUX_3[STRING[scratch_exp]bit](
  msb CONC msb CONC msb CONC s1[1..scratch_exp-3],
  msb CONC msb CONC msb CONC msb CONC s1[1..(scratch_exp-4)],
  msb CONC msb CONC msb CONC msb CONC msb CONC s1[1..scratch_exp-5],
  selector
).

```

```

#the carry to round, 1/2 value is rounded towards 0#

```

```

cs = CASE select
OF shift4: CASE msb
  OF b"1": s1[scratch_exp-3],      #neg no.#
  b"0": CASE s1[scratch_exp-3..scratch_exp]
    OF b"1000": b"0      #round down on 1/2 value#
    ELSE s1[scratch_exp-3]
    ESAC

```

```

    ESAC,
    shift3: CASE msb
      OF b'1': s1[scratch_exp-2], #neg no.#
      b'0': CASE s1[scratch_exp-2..scratch_exp]
        OF b'100': b'0 #round down on 1/2 value#
        ELSE s1[scratch_exp-2]
        ESAC
    ESAC,

    shift5: CASE msb
      OF b'1': s1[scratch_exp-4], #neg no.#
      b'0': CASE s1[scratch_exp-4..scratch_exp]
        OF b'10000': b'0 #round down on 1/2 value#
        ELSE s1[scratch_exp-4]
        ESAC
    ESAC,

    sum17 = ADD_US_ACTEL(B_TO_S cs, shift, b'1),
    sum = sum17[2..scratch_exp+1],
    #bit 1 is carry out, gives 16 bit sum#
    subsignal = (cs, sum),
    #ACTEL HACK#
    soa = CASE sum[1]
      OF b'1': 1, #saturate to -512#
      b'0': 1 #saturate to 512#
    ESAC,

    ss1 = CASE selector
      OF 1: CASE sum[4..7] #these are the 5 msb's form the 13 bit word#
        OF (b'1111' | b'0000'): 1 #value in range#

```

```

ELSE I
ESAC,

c: CASE sum[5..7]#these are the 3 msb's from the 12 bit word left after#
    # taking out the 4 sign extension bits#
    OF (b'111' | b'000'): 1    #value in range#
    ELSE I
    ESAC,

r: CASE sum[6..7]#these are the 2 msb's from the 11 bit word#
    OF (b'11' | b'00'): 1    #value in range#
    ELSE I
    ESAC,

out= MXT(STRING[scratch_exp-6]bit)(b'01111111111'b'1000000000',sum[7..scratch_exp],soa,1,ss1).
OUTPUT (S_TO_IN out)[2]
END.

MAC LINE_DELAY_ST(TYPE 1)=[(4):in,1_col:wr_address,1_col:rd_address,1_load:rw)
->
RAM[(4)?].
[4]!:

FN PR_ADDER_ST = (l_scratch:a b )
->
l_scratch:
(S_TO_(scratch_exp) ADD_S((l_TO_S(scratch_exp-1)a)[2],(l_TO_S(scratch_exp-1)b)[2])) [2].

FN ADD_SUB_ST = (l_scratch: a b, l_add:sel)
->

```

```
l_scratch:
BEGIN
```

```
  LET a_s=(l_TO_S[scratch_exp]a)[2],
      b_s=(l_TO_S[scratch_exp]b)[2],
      sel_bit = CAST(STRING(1)bit)sel.
```

```
#ACTEL#
```

```
  b_s_inv = XOR_B[scratch_exp](b_s, ALL_SAME[scratch_exp]sel_bit).
```

```
#cinb is active low so cast sel(add->0,sub->1) & invert it#
```

```
  out= ADD_S_ACTEL(a_s,b_s_inv,CAST(bit)INV(1)sel_bit ),
  binout=out[2..scratch_exp+1].
```

```
  OUTPUT (S_TO_l[scratch_exp]binout)[2]
END.
```

```
MAC ALL_SAME(INT n) = (STRING(1)bit:dummy)
```

```
->
```

```
STRING(n)bit:
```

```
BEGIN
```

```
  FAULT IF n < 1 THEN "N<1 in ALL_SAME" FI.
```

```
  OUTPUT IF n=1 THEN dummy
```

```
  ELSE dummy CONC ALL_SAME[n-1] dummy
```

```
FI
```

```
END.
```

```
MAC CAST [TYPE to] = (TYPE from:n)
```

```
->
```

```
to:ALIEN CAST.
```

```

MAC_ZERO[INT n] = (STRING[1bit:dummy])
->
STRING[n]bit:
BEGIN
  FAULT IF n < 1 THEN "N<1 In ZERO" FI.
  OUTPUT IF n=1 THEN b"0"
    ELSE b"0" CONC ZERO[n-1] b"0"
  FI
END.

MAC_B_TO_S = (bit:in)
->
  STRING[1]bit: CASE in
    OF b"0:b"0",
       b"1:b"1"
    ESAC.

MAC_I_TO_S[INT n] = (l_scratch: in)
->
  (flag, STRING[n]bit): BIOP_TRANSFORM_S.
MAC_S_TO_I[INT n] = (STRING[n]bit: in)
->
  (flag, l_scratch): BIOP_TRANSFORM_S.
MAC_S_TO_IN = (STRING[input_expbit: in]
->
  (flag, l_input): BIOP_TRANSFORM_S.
MAC_IN_TO_S[INT n] = (l_input: in)
->
  (flag, STRING[n]bit): BIOP_TRANSFORM_S.
MAC_U_TO_I[INT n] = (STRING[n]bit: in)
->

```

(flag,t_scratch): BIOP TRANSFORM_U.

MAC B_TO_1 = (bit:in)

->

t_scratch: CASE in

OF b'0:scratch/0,
b'1:scratch/1

ESAC.

MAC CARRY = (t_add:in)

->

STRING[1]bit: CASE in

OF add:b'0",
sub:b'1"

ESAC.

MAC BOOL_BIT = (bool:in)

->

STRING[1]bit:

CASE in

OF 1:b'1"

ELSE b'0"

ESAC.

MAC BIT_BOOL = (bit:in)

->

bool:

CASE in

OF b'1:

ELSE 1

ESAC.

```

MAC BOOL_STRING[INT n] = ((n)bool:in)
->
STRING[n] bit:
(LET out = BOOL_BIT in[1].
OUTPUT IF n=1
THEN out
ELSE out[1] CONC BOOL_STRING[n-1]((in[2..n]))
FI
).

#define a few useful gates #
FN NOT = (bool:in) -> bool:
CASE in
OF 1:1,
   1:1
ESAC.

FN MUX = (bool:sel in1 in2) -> bool:
# two input mux, select in1 if sel =1, otherwise in2 #
CASE sel
OF 1:in2,
   1:in1
ESAC.

FN XNOR=(bool in1 in2)->bool:
CASE (in1,in2)
OF (1,1)1,
   (1,1)1,
   (1,1)1,
   (1,1)1
ESAC.

```

```

FN XOR=(bool:in1 in2) ->bool:
CASE (in1,in2)
OF (f,f):f,
   (f,t):t,
   (t,f):f,
   (t,t):t
ESAC.

FN OR = (bool:in1 in2) ->bool:
CASE (in1,in2)
OF (f,f):(bool)((bool,t):t,
   (f,t):f
ESAC.

#FN MYLATCH = (bool:in) ->bool:DELAY(t,1).#

FN MYLATCH = (t_reset:reset,bool:in) ->bool:
BEGIN
MAKE PDEL[bool,0]:del.
LET out = CASE reset
OF rst:f
ELSE del
ESAC.
JOIN in->del.
OUTPUT out
END.

TYPE t_test = NEW(no/yes).
#.....#
#These functions change types from boolean to integer and vice-#
#versa. Supports 1 & 8 bit booleans.
#.....#

```



```

FN INT_BOOL=(l_input:k)    ->bool:      # 1bit input to binary #
CASE k
OF
  input/0:t,
  input/1:t
ESAC.

```

```

FN BOOL_INT=(bool:b) ->l_input:      # 1 bit bool to input #
CASE b
OF
  t:input/0,
  t:input/1
ESAC.

```

```

FN * =(l_input:a b)    ->l_input: ARITH a*b.
FN % =(l_input:a b)    ->l_input: ARITH a%b.
FN - =(l_input:a b)    ->l_input: ARITH a-b.
FN + =(l_input:a b)    ->l_input: ARITH a+b.
FN = =(l_input:a b)    ->l_test: ARITH IF a=b THEN 2 ELSE 1 FI.

```

```

COM
FN CHANGE_SIGN = (l_input:i) ->l_input:      #changes sign for 8-bit 2's#
  ARITH IF i<0 THEN 128+i      #complement no. #
  ELSE i
  FI.

```

```

FN SIGN = (l_input:i) ->bool:      #gets sign for 2's#
  ARITH IF i<0 THEN 1      #complement nos #
  ELSE 2
  FI.

```

```

FN TEST_SIZE = (l_input:x) ->bool:

```

```
#tests to see if the input is bigger than an 8-bit integer#
ARITH IF ( (x<=-128) AND (x>127)) THEN 1
ELSE 2 FI.
```

```
FN INT8_BOOL=(t_input:orig) ->{8}bool:
BEGIN
```

```
SEQ
VAR i1:=input/0, #input variables#
i0:=CHANGE_SIGN(orig),
b:=(1,1,1,1,1,SIGN(orig));
[INT n=1..7] (
i1:=i0%input/2;
b[n]:=INT_BOOL(i0-input/2n);
i0:=i1
);
OUTPUT CASE TEST_SIZE orig #checks to see if orig will#
OF t: [8]?bool, #fit input to an 8_bit value#
f: b
ESAC

END.
```

```
FN BOOL_INT8=([8]bool:b) ->{1}input: #converts 8bit boolean to 2's#
BEGIN
SEQ
VAR sum:=input/-128 * BOOL_INT([8]), #complement integer #
exp:=input/1;
[INT k=1..7] (
sum:=sum+exp * BOOL_INT(b[k]);
exp:=input/2 * exp
```

```

);
OUTPUT sum
END.

MOC
FN BOOL_INT10=((10)bool:b) ->l_input: #converts 10bit boolean to 2's#
BEGIN
    SEQ
        VAR sum:=input/-512 * BOOL_INT(b(10)). #complement integer #
            exp:=input/1;
            [INT k=1..9]
                (
                    sum:=sum+exp*BOOL_INT(b(k));
                    exp:=input/2 * exp
                );
            OUTPUT sum
        END.
    COM
    FN BOOL_INT16=((8)bool:in1 in2) ->l_input:
        # converts a 16-bit no., (fsbs,mabs) Input to Integer form)#
        (BOOL_INT8(in1))+((input/256)*BOOL_INT8(in2))+((input/256)*BOOL_INT(in1[8])).
        #hack because of sign extend#
        #of lsb #
    MOC
    #compute the mean square difference between two arrays of integers#
    FN MSE_COLOUR = (l_reset:reset,l_input:a b) ->[2]l_int32:
    BEGIN
        FN SAVE_ERROR = (l_reset:reset,l_int32:diff32) ->l_int32:
        BEGIN
            MAKE PDEL(l_int32,0)del,

```

```

PDEL(i_reset,0):edge.

LET rising = CASE (reset,edge)
  OF (no_reset,del):diff32,
     (no_reset,no_reset):del PL diff32
  ELSE del
  ESAC.

JOIN rising ->del,
reset ->edge.

OUTPUT del
END.

MAKE SAVE_ERROR:save_error.
LET out = (SEQ
  STATE VAR true_count INIT int32/1;
  VAR diff:=int32/0,
  diff32:=int32/0,
  incr:=int32/0;

  diff:=CASE reset
    OF reset:int32/0
    ELSE int32(a) MI int32(b)
    ESAC;
  incr:=CASE reset
    OF reset:int32/0
    ELSE int32/1
    ESAC;
  true_count:= CASE reset
    OF reset:int32/1
    ELSE true_count PL incr
    ESAC;

```

```
diff32:= (diff T1 diff);
```

```
OUTPUT (diff32,true_count) ).
```

```
JOIN      (reset,out[1])      -->save_error.  
OUTPUT    (save_error,save_error DV out[2])  
END.
```

#compute the mean square difference between two arrays of integers#

```
TYPE l_int32 = NEW int32/(-2147483000..2147483000).  
INT period_row=9.
```

```
FN l_32 = (l_input:in)  ->l_int32:ARITH in.
```

```
FN DV = (l_int32:a b)  ->l_int32:ARITH a%b.
```

```
FN PL = (l_int32:a b)  ->l_int32:ARITH a+b.
```

```
FN MI = (l_int32:a b)  ->l_int32:ARITH a-b.
```

```
FN TI = (l_int32:a b)  ->l_int32:ARITH a*b.
```

```
FN MSE_ROW = (l_input:a b) ->[3] l_int32:  
BEGIN SEQ
```

```
STATE VAR err INIT int32/0,
```

```
count INIT int32/0;
```

```
VAR diff:=int32/0,
```

```
diff32:=int32/0;
```

```
count:=count PL int32/1;
```

```

diff:=CASE count
  OF int32/(1..period_row).int32/0
  ELSE I_32(a) MI I_32(b)
  ESAC;
diff32:= (diff T1 diff);
err:=err PL diff32;
OUTPUT (err, err DV count,count)
END.

FN PRBS10 = (t_reset:reset) ->{10}bool:
#A 10 bit prbs generator,feedback taps on regs 3 & 10.#
BEGIN
  MAKE {10}MYLATCH1,
  XNOR:xnor.

  FOR INT k=1..9 JOIN
    (reset,[k]) ->{k+1}.

  JOIN (reset,xnor) ->{1},
    ({10},[3]) ->xnor.

  OUTPUT 1
END.

FN PRBS11 = (t_reset:reset) ->{11}bool:
#A 11 bit prbs generator,feedback taps on regs 2 & 11.#
BEGIN
  MAKE {11}MYLATCH1,
  XNOR:xnor.

```

```

FOR INT k=1..10 JOIN
  (reset,[k]) ->[k+1].
JOIN (reset,xnor) ->[1].
      ([1],[2]) ->xnor.

OUTPUT [1..10]

END.
COM
FN PRBS16 = (bool:reset) ->[16]bool:
#A 16 bit prbs generator,feedback taps on regs 1,3,12,16#
BEGIN
  MAKE [16]MYLATCH:l,
  XOR_4:xor,
  NOT:xnor.

FOR INT k=1..15 JOIN
  (ck,reset,[k]) ->[k+1].
JOIN (ck,reset,xnor) ->[1].
      ([1],[3],[16],[12]) ->xor,
      xor ->xnor.
OUTPUT ([INT k=1..16][k])

END.
FN PRBS12 = (clock:ck,bool:reset) ->[12]bool:
#A 12 bit prbs generator,feedback taps on regs 1,4,6,12.#
BEGIN
  MAKE [12]MYLATCH:l,
  XOR_4:xor,
  NOT:xnor.

```

```

FOR INT k=1..11 JOIN
  (ck,reset,[k]) ->[k+1].

JOIN (ck,reset,xnor) ->[1].
  ([1],[4],[6],[12]) ->xor,
  xor ->xnor.
OUTPUT ((INT k=1..12)[k])
END.

FN PRBS8 = (clock:ck,bool:reset) ->[8]bool:
#A 8 bit prbs generator,feedback taps on regs 2,3,4,8.#
BEGIN
  MAKE [8]MYLATCH1,
  XOR_4:xor,
  NOT:xnor.

FOR INT k=1..7 JOIN
  (ck,reset,[k]) ->[k+1].

JOIN (ck,reset,xnor) ->[1],
  ([2],[3],[4],[8]) ->xor,
  xor ->xnor.
OUTPUT ((INT k=1..8)[k])
END.

MOC
#TEST FOR Y U V #
#to test the 2d convolver using prbs input into the forward convolver#
#then outputting to the inverse convolver and checking against the original result#

```


- 537 -

```

prbs      ->int_bool,
(ck,reset,int_bool,extwrite_in,csel_in,reg_sel,value)  ->dw1.

#calculate the mse error for each channel#
FOR INT j=1..3 JOIN
(CASE dw1[2][j])
OF read:rst
ELSE no_rst
ESAC,dw1[1][int_bool] ->mse_colour[j].
OUTPUT (mse_colour[1][1],mse_colour[2][1],mse_colour[3][1])
END.

FN DWT = (bool,t_reset,t_input,bool,bool,t_sparc_addr:reg_sel value)  ->(t_input,[3]t_load,[3]t_load):IMPORT.
MAC PDEL(TYPE t,INT n) = (t) ->:IMPORT.

IMPORTS      dw1/sing: DWT_TEST( RENAMED DWT) PDEL.

#TEST FOR LUMINANCE ONLY#
#to test the 2d convolver using prbs input into the forward convolver#
#then outputting to the inverse convolver and checking against the original results#

FN TEST_Y = (bool:ck,t_reset:reset,bool:extwrite_in csel_in, t_sparc_addr:reg_sel value,t_reset:prbs_reset)
->[2]t_int32:

BEGIN
  FN DEL = (t_load:in)  ->t_load:DELAY(read,1).
  FN PULSE = (t_load:in) ->t_reset:
  CASE (in,DEL in)

```

```

OF (write_read):rst
ELSE no_rst
ESAC.

MAKE PRBS11:prbs,
BOOL_INT10:int_bool,
DWT:dwt,
MSE_COLOUR:mse_colour.

JOIN (CASE (prbs_reset,PULSE dwt[2][1]) #run the prbs at start, or on out of IDWT#
OF (rst,1_reset))((1_reset,rst):rst
ELSE no_rst
ESAC)
->prbs,

prbs
->int_bool,
(ck_reset,int_bool,extwrite_in,cal_in,reg_sel,value)
->dwt,
(CASE dwt[2][1]
OF read:rst
ELSE no_rst
ESAC,dwt[1],int_bool) ->mse_colour.

OUTPUT mse_colour
END.

```

APPENDIX B-2

#test for abs #

FN ABS_TEST = (STRING[10]bit:in in2) ->bool: in LE_U in2.
 #the state machine to control the address counters#
 #only works for 3 octave decomposition in y/2 in ulv#

FN CONTROL_ENABLE = (bool:ck,t reset:reset,t channel:new_channel channel,[3]bootc_blk,STRING[2]bit:subband,
 t_load:load_channel,t_mode:new_mode)

->([3]bool#en_blk#,t_oclave,[2]bool#tree_done,tf_block_done?,t_state#reset_state#):

BEGIN

MAKE DF1(t_state):state.

#set up initial state thro mux on reset, on HH stay in zz0 state#

LET start_state = CASE channel

OF ulv:down t,

y:up0

ESAC,

reset_state = CASE reset

OF rst: start_state

ELSE state

ESAC.

LET next_values = (SEQ

VAR en_blk:=13j!, #enable blk count#

tf_block_done:=1, #enable x_count for LPF#

tree_done:=1, #enable x_count for other subbands#

new_state:=reset_state,

oclave:=?t_oclave; #current octave#

CASE reset_state

```

ELSE
  ESAC),
  zz1: ( octave:=oct0;
    en blk1:=t;
    CASE c_blk1
  OF  t(new_state:=zz2;
    en blk2:=0)
  ELSE
    ESAC),
  zz2: ( octave:=oct0;
    en blk1:=t;
    CASE c_blk1
  OF  t(new_state:=zz3;
    en blk2:=0)
  ELSE
    ESAC),
  zz3: ( octave:=oct0;
    en blk1:=t;
    #now decide the next state, on block(1) carry check the other block carries#
    CASE c_blk1
  OF  t(new_state:=down1;
    en blk2:=t; #roll over to 0#
    en blk3:=t   #because state zz3 clock 1 pulse#
    )
  ELSE
    ESAC
  ),
  down1: ( octave:=oct1;
    en blk2:=t;
    CASE o_blk2

```

```

OFup0: ( octave:=oct/2;
        en_blk(3):=t;
        CASE c_blk(3)
        OF t(CASE subband
            OF b'00':t(1) block_done:=1 #clock x_count for LPF y channel#
            ELSE new_state:=up1 #change state when count done#
            ESAC;

CASE new_mode #in luminance & done with that tree#
OF stop:tree_done:=1
   ELSE
   ESAC)
   ELSE
   ESAC);
up1: ( octave:=oct/1;
      en_blk(2):=t;
      CASE c_blk(2)
      OF t(new_state=zz0;

CASE new_mode #in luminance, terminate branch & move to next branch#
OF stop:(new_state:=down1;
          en_blk(3):=1)
   ELSE
   ESAC)
   ELSE
   ESAC);
zz0: ( octave:=oct/0;
      en_blk(1):=t;
      CASE c_blk(1)
      OF t(new_state=zz1;
          en_blk(2):=1)

```

```

OF t:(CASE subband
  OF b"00":lpf_block_done:=t    #clock x_count for LPF u/v channel#
  ELSE new_state:=zz0    #change state when count done#
ESAC;

CASE (new_mode,channel)    #stop so finish this tree/branch & move on#
OF (stop,u/v):tree_done:=t,
  (stop,y):(en_blk[3]:=t;
    CASE c_blk[3]    #move to next tree#
    OF tree_done:=t
    ELSE new_state:=down1
    ESAC
  )
ELSE
ESAC)
ELSE
ESAC)
ELSE
ESAC)

ESAC;

CASE channel
OF u/v: CASE (c_blk[1],c_blk[2])
  OF (t,t):tree_done:=t
  ELSE
ESAC,
  y: CASE (c_blk[1],c_blk[2],c_blk[3])
  OF (t,t,t):tree_done:=t
  ELSE
ESAC

ESAC;

```



```

#now change to start state if the sequence has finished#
CASE tree_done #in LPF state doesn't change when block done#
OF: new_state:= start_state
ELSE
ESAC;
#on channel change, use starting state for new channel#
CASE load_channel #in LPF state doesn't change when block done#
OF write: new_state:= CASE new_channel
OFy: up0,
ujv: down1
ESAC
ELSE
ESAC;

```

```

OUTPUT (new_state, en_blk, octave, (tree_done, lpf_block_done))
).

```

```

JOIN (ck, next_values[1]) -> state.
OUTPUT (next_values[2], next_values[3], next_values[4], reset_state)
END.

```

```

FN CHECK = (t_input < sub size y, t_octave: oct) -> l_sparc_addr:
ARITH ((x SL 1) + (1 AND sub) + size*(y SL 1) + (sub SR 1))) SL oct.

```

```

#these are the addr gens for the x & y addresses of a pixel given the octave#
#sub&blk no. for each octave. Each x&y address is of the form #
# x= count(5 bits)(blk(3)..blk(octave+1)))(s){octave 0's} #

```

```

# y= count(5 bits){blk(3)..blk(octave+1)}{s} {octave 0's} #
#this makes up the 9 bit address for CIF Images #
#the blk & s counters are vertical 2 bit with the lsb in the x coord #
#and carry out on 3, last counter is both horiz and vertical counter #
#read_enable enable the block count for the read address, but not the #
#carry-outs for the mode change, this is done on the write addr cycle #
#by write_enable, so same address values generated on read & write cycles#

FN ADDR_GEN = (bool:ck, t_reset:reset, t_channel:new channel channel, load:load_channel, STRING[2]bit:sub_count,
STRING[ysize]bit:col_length, STRING[ysize]bit:row_length, STRING[ysize]bit:image_string,
STRING[ysize]bit:image_string, STRING[1]bit:image_string_3, image*2.5#,
bool:read_enable write_enable, t_mode:new_mode)

-> (t_spare_addr, t_octave, bool#sub_finished#, bool#tree_done#, bool#lpf_done#, t_state):
BEGIN
  MAKE COUNTER{ysize-4}:x_count,
  COUNTER{ysize-4}:y_count,
  CONTROL_ENABLE:control,
  [3]BLK_SUB_COUNT:blk_count.

#size of lpf images/2 -1, for y, u, v. /2 because count in pairs of lpf values #
#lpf same size for all channels!!!#

LET (x_lpf, y_lpf) = (col_length[1..ysize-4], row_length[1..ysize-4]).

tree_done = control[3][1],
lpf_block_done = control[3][2],
x_en = CASE (tree_done, lpf_block_done)
  OF (t, bool){(bool, t):
  ELSE f
  ESAC,

```

```
blk_en=control[1],
oclave=control[2],
```

#clk_y_count when all blocks done for subs 1-3, or when final blk done for lpf#

```
y_en = CASE sub_count
OF b'00': CASE (lpf_block_done, x_count[2])
  OF (f, f):
    ELSE f
  ESAC
ELSE CASE (iree_done, x_count[2])
  OF (f, f):
    ELSE f
  ESAC
ESAC,
```

```
x_msb_out = CASE channel
OF y: x_count[1] CONC B TO S(blk_count[3][1][2]), #always the msb bits#
  ulv: b'0' CONC x_count[1]
ESAC,
y_msb_out = CASE channel
OF y: y_count[1] CONC B TO S(blk_count[3][1][1]),
  ulv: b'0' CONC y_count[1]
ESAC,
```

```
x_lsb_out = CASE (octave) #bit2 is lsb#
OF (oct0): ((INT k=1..2) blk_count[3-k][1][2]) CONC sub_count[2],
  (oct1): (blk_count[2][1][2], sub_count[2], b'0),
  (oct2): sub_count[2] CONC [2]b'0
ESAC,
```

- 547 -

```

y_lsb_out = CASE (octave) #bit 1 is msb#
OF (oct0):(INT k=1..2)blk_count[3-k][1]CONC sub_count[1],
  (oct/1):(blk_count[2][1][1], sub_count[1], b'0),
  (oct/2):sub_count[1]CONC [2]b'0
ESAC,
x_addr = x_msb_out CONC BIT_STRING(3)x_lsb_out,
y_addr = y_msb_out CONC BIT_STRING(3)y_lsb_out,

#enable the sub band counter#
sub_en = CASE (y_count[2]y_en)
OF (1):1
ELSE 1
ESAC,

!pf_done = CASE sub_count
OF b'00':sub_en #!!!!CHANGE ACCORDING TO LATENCY IN DECODE#
ELSE 1
ESAC,

base_y_sel = CASE channel
OF y1,
  u:c,
  v:r
ESAC,

base_rows = MUX_3(STRING(1)1b1)(ZERO(1)1b'0'.b'0' CONC yimage_string[1..yatz]CONC b'0',
  yimage_string_3base_y_sel,
#base address for no of rows for y,u & v memory areas#

address = x_addr ADD_U ((y_addr ADD_U base_rows)[2..12]) MULT_U (CASE channel
  OF y:image_string,

```

```

        ulv(SR_U(1)ximage_string)[1..xsize]
        ESAC)
    ),
    int_addr = (S_TO_SPARC address)[2].
    JOIN (ck,reset,x_en,x_lpf)      ->x_count,
          (ck,reset,y_en,y_lpf)      ->y_count,
    #use new_channel so on channel change control state picks up correct value#
    (ck,reset,new_channel,channel,((INT k=1..3)blk_count[[2]],sub_count,load_channel,new_mode)
    ->control.
    FOR INT k=1..3 JOIN (ck,reset,blk_en[k],read_enable OR write_enable,write_enable) ->blk_count[k].

    OUTPUT (int_addr,oclave,sub_en,tree_done,lpf_done,control[4])
    END.
    ,
    #a counter to control the sequencing of r/w, token, huffman cycles#
    #decide reset is enabled 1 cycle early, and latched to avoid glitches#
    #lpf_stop is a dummy mode to disable the block writes&huffman data#
    #cycles for that block#
    FN CONTROL_COUNTER = (bool:ck.l_reset:reset.l_mode:mode new_mode.l_direction:direction)
    ->(l_load,l_cycle,l_reset,bool:bool,l_load,l_cs,l_load,l_cs):

    #mode load,cycle,decide reset,read_addr_enable,write_addr_enable,load flags#
    #decode write_addr_enable early and latch to avoid feedback loop with pro_mode#
    #in MODE_CONTROL#
    BEGIN
        MAKE COUNT_SYNC[4]:count.

```

```
LET count_len = (U_TO_LEN(4) count[1])[2].
```

```
LET out = (SEQ
```

```
VAR cycle:=skip_cycle,
    decide_reset:=no_rst,
    load_mode:=read,
    load_flags:=read,
    cs_new:=no_select,
    cs_old:=select,
    rw_old:=read,
    read_addr_enable:=f,
    write_addr_enable:=f;
```

```
CASE direction
```

```
OF forward: CASE mode
```

```
OF send|still_send|pf_send: CASE count_len
    OF len/(0..3):(read_addr_enable:=f;
        cs_new:=select),
    len/(4):(cycle:=token_cycle;
        load_flags:=write;
        write_addr_enable:=f),
```

```
len/(5..7):(write_addr_enable:=f;
```

```
    CASE new_mode
```

```
    OF stop|pf_stop:(cycle:=skip_cycle;
```

```
        rw_old:=read;
```

```
        cs_old:=no_select),
```

```
    vold:(cycle:=skip_cycle;
```

```
        rw_old:=write)
```

```
    ELSE (cycle:=data_cycle;
```

```
        rw_old:=write)
```

```

    ESAC),
len/8:(decide_reset:=rst;
CASE new_mode
  OF stop|prf_stop:(cycle:=skip_cycle;
    rw_old:=read;
    cs_old:=no_select),
    void:(cycle:=skip_cycle;
      load_mode:=write;
      rw_old:=write)
  ELSE (cycle:=data_cycle;
    load_mode:=write;
    rw_old:=write)
    ESAC)
ELSE
ESAC,

still: CASE count len
  OF len/(0..3):(read_addr_enable:=t;
    cs_new:=select),
    len/(4):(cycle:=token_cycle;
      write_addr_enable:=t;
      load_flags:=write),
    len/(5..7):(rw_old:=write;
      write_addr_enable:=t;
      CASE new_mode
        OF void_still:(cycle:=skip_cycle
        ELSE cycle:=data_cycle
        ESAC),
len/8:(decide_reset:=rst;

```

```

rw_old:=write;
load_mode:=write;
CASE new_mode
OF  void_still:cycle:=skip_cycle
ELSE cycle:=data_cycle
ESAC)

```

```

ELSE
ESAC,

```

```

hpf_still:
CASE count_len
OF  len/(0..3):(read_addr_enable:=t;
    cs_new:=select);
len/(4):(cycle:=token_cycle;
    write_addr_enable:=t;
    load_flags:=write);
len/(5..7):(cycle:=data_cycle;
    rw_old:=write;
    write_addr_enable:=f);
len/8:(cycle:=data_cycle;
    rw_old:=write;
    decide_reset:=rst;
    load_mode:=write)

```

```

ELSE
ESAC,

```

```

void:
CASE count_len
OF  len/(0..3):(read_addr_enable:=t;
    cs_new:=select);
len/4:(load_flags:=write;
    cycle:=token_cycle; #dummy token cycle for mode update#

```



```

write_addr_enable:=1),
  len/5..7:(write_addr_enable:=1; #keep counters going#
CASE new_mode
  OF
    stop:(rw_old:=read;
           cs_old:=no_select)
  ELSE rw_old:=write
  ESAC),
  len/8:(decide_reset:=rst;
CASE new_mode
  OF
    stop:(rw_old:=read;
           cs_old:=no_select)
  ELSE (load_mode:=write;
        rw_old:=write)
  ESAC)

```

```

ELSE
ESAC,

```

```

void_still: CASE count_len
  OF len/0: write_addr_enable:=1, #allow for delay#
  len/1..3:(write_addr_enable:=1;
            rw_old:=write),
  len/4:(rw_old:=write;
         load_mode:=write;
         decide_reset:=rst)
  ELSE
  ESAC

```

```

ELSE
ESAC,

```

```

inverse: CASE mode

```

```

OF sendstill_send|pf_send: CASE count_len
  OF len/(0..3):(read_addr_enable:=f),
    len/(4):(cycle:=token_cycle;
      write_addr_enable:=t;
      load_flags:=write);
    len/(5..7):(write_addr_enable:=t;
      CASE new_mode
        OF stop|pf_stop:(cycle:=skip_cycle;
          rw_old:=read;
            cs_old:=no_select),
          voldt(cycle:=skip_cycle;
            rw_old:=write)
        ELSE (cycle:=data_cycle;
          rw_old:=write)
          ESAC);
    len/8:(decide_reset:=rsi;
      CASE new_mode
        OF stop|pf_stop:(cycle:=skip_cycle;
          rw_old:=read;
            cs_old:=no_select),
          voldt(cycle:=skip_cycle;
            load_mode:=write;
            rw_old:=write)
          ESAC)
        ELSE (cycle:=data_cycle;
          load_mode:=write;
          rw_old:=write)
          ESAC)
      ELSE
        ESAC,

```

```

still:  CASE count_len
        OF len(0);,      #skip to allow reset in Huffman#
        len(1):(cycle:=token_cycle;
            write_addr_enable:=1),
        len(2..4):(rw_old:=write;
            write_addr_enable:=1;
        CASE new_mode
        OF void_still:cycle:=skip_cycle
        ELSE cycle:=data_cycle
        ESAC),

        len(5):(rw_old:=write;
            decide_reset:=rst;
            load_mode:=write;
        CASE new_mode
        OF void_still:cycle:=skip_cycle
        ELSE cycle:=data_cycle
        ESAC)
        ELSE
        ESAC,
    bpf_still: CASE count_len
        OF len(0);,      #match with previous#
        len(1):(        #skip for write enable delay#
            write_addr_enable:=1,
            len(2..4):(cycle:=data_cycle;
                rw_old:=write;
                write_addr_enable:=1),
            len(5):(cycle:=data_cycle;
                rw_old:=write;
                decide_reset:=rst;
                load_mode:=write)

```

```

ELSE
ESAC,
CASE count_len
OF len/(0..3):(read_addr_enable:=1),
   len/4:(load_flags:=write;
   cycle:=token_cycle; #dummy token cycle for mode update#
   write_addr_enable:=1),
   len/(5..7):(write_addr_enable:=1;
CASE new_mode
OF stop:(rw_old:=read;
   cs_old:=no_select)
   ELSE rw_old:=write
   ESAC),
len/8:(decide_reset:=rst;
CASE new_mode
OF stop:(rw_old:=read;
   cs_old:=no_select)
   ELSE (load_mode:=write;
   rw_old:=write)
   ESAC)

ELSE
ESAC,
CASE count_len
OF len/(0);, #match with rest#
   len/1:(write_addr_enable:=1, #dummy as write delayed#
   len/(2..4):(write_addr_enable:=1;
   rw_old:=write),
   len/5:(rw_old:=write;
   load_mode:=write;
   decide_reset:=rst)
   ELSE

```

- 556 -

ESAC

ELSE
ESAC

ESAC;

```

OUTPUT (load_mode,cycle,DF1(t_reset))(ck,decide_reset),read_addr_enable,
DFF{boof}(ck,reset,write_addr_enable,t,load_flags,
cs_new,rw_old,cs_old)
    ).

```

```

JOIN (ck,CASE reset
OF rst:rst
ELSE out[3]
ESAC,t) ->countL

```

```

OUTPUT out
END.

```

```

# .....#
# A set of boolean ,le gate level counters
# .....#

```

```

# .....#
# The basic toggle flip-flop plus and gate for a synchronous counter
#input t is the toggle ,outputs are q and tc (toggle for next counter)
#stage
# .....#

```

```

MAC BASIC_COUNT = (bool:ck ,t_reset:reset,boof:log) ->(STRING[1]boof,boof):

```

- 557 -

```

BEGIN
    MAKE OFF(bool):dlat,
    XOR :xor,
    AND :and.

    JOIN (ck,reset,xor,1)->dlat,
        (dlat,log) ->and,
        (log,dlat) ->xor.
    OUTPUT (CAST(STRING(1)bit) dlat,and)
END.

#.....#
# The n-bit macro counter generator, en is the enable, the outputs #
# are msb(bit 1).....lsb,carry. This is the same order as ELLA strings are stored#
#.....#

MAC COUNT_SYNC(INT n) = (bool:ck,1_reset: reset,bool: en )->(STRING(n)bit,bool):
(LET out = BASIC_COUNT(ck,reset,en) .

    OUTPUT ( IF n=1
        THEN (out[1],out[2])
        ELSE ( LET outn = COUNT_SYNC(n-1)(ck,reset,out[2]) .
            OUTPUT (outn[1] CONC out[1],outn[2])
        )
        FI)
    ).
COM
FN TEST_COUNT_SYNC = (bool:ck,1_reset: reset,bool: en ) -> ([4]bool,bool):
COUNT_SYNC(4)(ck,reset,en).
MOC

```

```

#.....#
#The basic toggle flip-flop plus and gate for a synchronous counter #
#input t is the toggle, updown delms the direction ,outputs are q and #
# tc (toggle for next counterstage, active low for down/high for up) #
#.....#

MAC BASIC_COUNT_UD = (boot:ck !_reset:reset,boot:log, t_updown:updown) ->[2]bool:

BEGIN
    MAKE DFF{boot}:dlat.
    LET toggle = tog.
    xorn = CASE updown
    OF up: CASE (toggle,dlat) #xor#
    OF (t,1)|(f,0):f
    ELSE t
    ESAC.
    down:CASE (toggle,dlat) #xnor#
    OF (t,1)|(f,0):f
    ELSE f
    ESAC
    ESAC.
    cout = CASE updown
    OF up:CASE (dlat,toggle) #AND#
    OF (t,1):f
    ELSE f
    ESAC.
    down:CASE (dlat,toggle) #OR#
    OF (f,0):f
    ELSE t
    ESAC

```

ESAC.

```
JOIN (ck,reset,xorn,n)->dlat.
OUTPUT (dlat,count)
```

END.

```
#.....#
# The n-bit macro u/d counter generator, en is the enable, the outputs #
# are msb(bit 1).....lsb,carry. This is the same order as ELLA strings are stored#
# first enable is active low on down, so invert. #
#.....#
MAC COUNT_SYNC_UD[INT n] = (bool:ck,t_reset: reset,bool:en,t_updown:updown) ->(STRING[n]bit,bool):
BEGIN
  MAKE [n]BASIC_COUNT_UD:basic_count.
  LET enable = ((INT k=1..n-1) basic_count[k+1][2]) CONC CASE updown #invert enable if down count#
  OF up:en
  ELSE NOT en
  ESAC.
  FOR INT k=1..n JOIN (ck,reset,enable[k],updown) ->basic_count[k].
  OUTPUT (BOOL_STRING[n]((INT k=1..n)basic_count[k][1]), basic_count[1][2])
END.
```

COM

```
FN TEST_COUNT_SYNC_UD = (bool:ck,t_reset: reset,bool:en,t_updown:updown) ->([4]bool,bool):
COUNT_SYNC_UD[4](ck,reset,en,updown).
```

MOC

```
#the basic x/y counter, carry out 1 cycle before final count given by x_lpf/y_lpf#
MAC COUNTER[INT n] = (bool:ck,t_reset:reset,bool:en,STRING[n]bitx_lpf) ->(STRING[n]bit,bool):
BEGIN
```



```

MAKE COUNT_SYNC(n):x_count.

LET out = x_count{1}.
final_count = out EQ_U x_lpf,
final_count_en = CASE (final_count, en)
  OF (1):1
  ELSE f
  ESAC,
#reset after 4 counts at final count value#
cnt_reset = CASE reset
  OF rst:rst
  ELSE CASE DF1(boot)(ck, final_count_en) #reset taken out of DFF 12/6#
    OF 1rst
    ELSE no_rst
    ESAC
  ESAC.
JOIN (ck, cnt_reset, en) -> x_count.
OUTPUT (out, final_count)
END.
COM
#the basic y counter, carry out 1 cycle before final count given by y_lpf#
#reset at end of channel given by system reset #
MAC Y_COUNTER = (boot, ck, 1, reset, reset, boot, en, STRING(4)bit_y_lpf) -> (STRING(4)bit, boot):
BEGIN
MAKE COUNT_SYNC(4):y_count.

LET out = y_count{1}.
JOIN (ck, reset, en) -> y_count.
OUTPUT (out, out EQ_U y_lpf)

```

END.
MOC

```
COM
#the blk, or sub-band counters, carry out on 3#
FN BLK_SUB_COUNT = (bool:ck,l_reset:reset, bool:en)    ->(STRING[2]blk,bool):
BEGIN
  MAKE COUNT_SYNC[2]:blk_count.
  LET out = blk_count[1].
  JOIN (ck,reset,en) ->blk_count.
  OUTPUT(out,out EQ_U (C_TO_S[2]cd/3)[2])
END.
MOC
```

#the blk, or sub-band counters, carry out on 3, cout_en enables the carry out, & cin_en AND en enables the count#

```
FN BLK_SUB_COUNT = (bool:ck,l_reset:reset, bool:en cin_en cout_en)    ->(STRING[2]blk,bool):
```

```
BEGIN
  MAKE COUNT_SYNC[2]:blk_count.
  LET out = blk_count[1].
  JOIN (ck,reset,en AND cin_en)    ->blk_count.
  OUTPUT(out,(out EQ_U (C_TO_S[2]cd/3)[2]) AND cout_en)
END.
```

```
FN LAST_BLK_COUNT = (bool:ck,l_reset:reset, bool:en,l_channel:channel,bool:line_finished) ->
  (STRING[2]blk,[2]bool#x_en,y_end):
BEGIN
  MAKE BASIC_COUNT : lsb_msb.
  JOIN (ck,reset,en) ->lsb,
  (ck,reset, CASE channel
```

```

OF y:lsb[2],
u/v:line_finished
ESAC) ->msb.

LET out = (msb[1])CONC(msb[1]).
OUTPUT (out, CASE channel
OF y:(out EQ U (C TO S[2]cod[3])[2].line_finished),
u/v:(lsb[2].msb[2])
ESAC)
END.

#the L1 norm calculator/ comparison constants & flag values#
#adding 4 absolute data values so result can grow by 2 bits#
#5 cycle sequence, a reset cycle with no data input, followed#
#by 4 data cycles#

MAC L1NORM = (bool:ck, t_reset:reset, STRING[(INT n)bit:in] ->STRING[n+2]bit:
BEGIN
MAKE DF1(STRING[n+4]bit:in2.

LET in_s = in,
msb = ALL_SAME(n)(B TO Sin_s[1]).
COM
add_in1 = ln2 CONC in_s[1]. #in_s[1] is the carry in to the adder#
#lsb so gen carry to next bit#
add_in2 = ((in_s XOR_B msb)CONC in_s[1]).
#adder=ADD_U(add_in1,add_in2),#
MOC
add_in1 = (in_s XOR_B msb).
rst_mux = CASE reset
OF rst:ZERO(n+4)b'0"
ELSE in2

```

ESAC,

```

adder=ADD_US_ACTEL(add_in1,rst_mux,CASE in_s[1]
  OF b'1:b'0
  ELSE b'1
  ESAC),
out =adder[2..(n+5)].

```

JOIN (ck,out) ->in2.

OUTPUT in2[3..n+4]
END.

#the block to decide if all its inputs are all 0#

```

FN ALL_ZERO = (bool:ck, t_reset:reset, t_input:in) ->bool:
BEGIN

```

MAKE DF1{bool}:out.

LET in_s = (IN_TO_S(input_exp)[n])[2].

```

in_eq_0 = in_s EQ_U ZERO(input_exp)b'0'; #in -0#
#1 if reset high, & OR with previous flag#
all_eq_0 = CASE reset
  OF rst: in_eq_0
  ELSE CASE out
    OF tf
    ELSE in_eq_0
  ESAC
  ESAC.

```

```

JOIN (ck,all_eq_0)->out.
OUTPUT out
END.

```

```

MAC ABS_NORM = (bool:ck, l_reset:reset, STRING(result_exp-2)bit:qshift, STRING(int_n)bit:n)
->(STRING(n+2)bit,bool:all < qshift);

```

```

BEGIN
  MAKE DF1(STRING(n+4)bit):in2,
  DF1{bool}:out.
  LET abs_in = ABS_S in,
  rst_mux = CASE reset
    OF rst:ZERO(n+4)b'0"
    ELSE in2
    ESAC,

```

```

  adder = ADD_US_ACTEL(abs_in,rst_mux,b'1),
  add_s = adder[2..(n+5)],
  in_small = abs_in LT_U qshift,
  #1 if reset high, & OR with previous flag#
  all_small = CASE reset
    OF rst: 1
    ELSE CASE in_small
    OF hf
    ELSE out
    ESAC
  ESAC.

```

```

JOIN (ck,add_s) ->in2,
(ck,all_small) ->out.

```

```

OUTPUT (in2[3..n+4],out)

```

- 565 -

END.

```

#the decide fn block#
FN DECIDE = (bool:ok,t_reset:reset,t_result:q_hrt,t_input:new old,t_result: threshold comparison,
             t_octave:ocds,t_load:load_flags) ->[7]bool:
#nzflag,origin,noflag,ozflag,moflon,pro_new_z,pro_no_z#
BEGIN
    MAKELINORM(input_exp): oz,
    ABS_NORM(input_exp): nz,
    ABS_NORM(input_exp+1):no,
    LATCH([7]bool):flags.

LET qshift=(l TO SC(result_exp)q_hrt)[2][1..result_exp-2].
#divide by 4 as test is on coeff values not block values#

n_o=(lN TO S(input_exp)new)[2] SUB_S (lN TO S(input_exp)old)[2], #new-old,use from quant#
nzflag = nz[1] LE_U (l TO SC(result_exp)threshold)[2], #delay tests for pipelined data#
noflag = no[1] LE_U (l TO SC(result_exp)comparison)[2],
ozflag = oz EQ_U ZERO(input_exp)b'0',
origin = nz[1] LE_U no[1],
nz_plus_oz = nz[1] ADD_U oz,

pro_new_z = nz[2],

pro_no_z = no[2],

shift_add_sel = CASE DF (t_octave)(ck,ocds) #delay ocds to match pipelin delay#
OF ocd/oruno,

```

- 566 -

```

oct/1: dos,
oct/2: tres,
oct/3: quatre
    ESAC,
#keep 13 bits here to match no, keep msb's#
    shift_add = MUX_4(STRING(input_exp+3)bit) (      #delay octs to match pipelin delay#
        nz_plus_oz[1..input_exp+3],
        b'0'CONC nz_plus_oz[1..input_exp+2],
        b'00'CONC nz_plus_oz[1..input_exp+1],
        b'000'CONC nz_plus_oz[1..input_exp],
        shift_add_sel
    ),

    motion = shift_add LE_U no{1},
    #value for simulation#
    nz_r = (SC_TO_I(12) nz{1})[2],
    no_r = (SC_TO_I(13) no{1})[2],
    oz_r = (SC_TO_I(12) oz)[2],
    sa_r = (SC_TO_I(13) shift_add)[2].

JOIN (ck,reset,qshift,(IN_TO_S(input_exp)new)[2]) ->nz,
      (load_flags,(nzflag,origln,noflag,ozflag,motion,pro_new_z,pro_no_z)) ->flags,
      (ck,reset,qshift,CAST(STRING(input_exp+1)bit)n_o) ->no,
      (ck,reset,(IN_TO_S(input_exp)old)[2]) ->oz.
OUTPUT flags
END.
#the buffer for the FIFO#

#a pulse generator, glitch free#
FN PULSE = (bool:ck,reset,load.in) ->{load:

```

- 567 -

```

CASE (in_DFF(i_load)(ck,reset,in,read))
OF (write,read):write
ELSE read
ESAC.

```

```

#the length of the huffman encoded word#
FN LENGTH = (i_input:mag_out) ->STRING(5)bit:
CASE mag_out #length of inputoded word#
OF input/0:b"00001",
input/1:b"00011",
input/2:b"00100",
input/3:b"00101",
input/4:b"00110",
input/5:b"00111",
input/6:b"01000",
input/(7..21):b"01100"
ELSE b"10000"
# input/(22..37):b"10000"#
ESAC.

```

```

FN REV_BITS = (STRING(8)bit:in) ->STRING(8)bit:CAST(STRING(8)bit)(in[8],in[7],in[6],in[5],in[4],in[3],in[2],in[1]).

```

```

FN FIFO_BUFFER = (bool:ck, i_resatreset, i_direction:direction, i_cycle:cycle, i_mode:mode,
i_input:value mag_out huff, STRING(16)bit:fifo_in, i_fifo:fifo_full fifo_empty,
STRING(32)bit:shift,STRING(2)bit:token_length, bool:flush_buffer, i_quant:prf_quant)

```

```

->(STRING(16)bit,STRING(16)bit,STRING(16)bit,STRING(5)bit, i_load, i_load):
#fifo_out, s, fifo_read fifo_write#

```

```

BEGIN
MAKEDFF_INIT(STRING(16)bit):low_word high_word,

```



```

DFF_INIT(STRING[5bit]:s,
DFF_INIT(1,high_low):high_low,
MUX_2(STRING[16bit]:high_in_low_in,high_out_low_out,

```

```

LET dir_sel = CASE direction
  OF forward:left
  ELSE right
  ESAC,

```

```

length = CASE cycle
  OF token_cycle:b'000' CONC token_length,
     skip_cycle:b'00000',
     data_cycle: CASE mode #on LPF STILL length fixed, given by input_exp-shift const#
       OF 1st_still:((LEN_TO_U(5) len/input_exp)[2] SUB_U
         (Q_TO_U(3) 1st_quant)[2])(2..6)
         ELSE LENGTH_MUX_2(1, input)(value,mag_out_huff_dir_sel)
       ESAC
  ESAC,

```

```

selected_s = CASE direction
  OF forward:b'0' CONC s[2..5]
  ELSE s
  ESAC,

```

```

new_s = (ADD_US_ACTEL(selected_s,length,b'1'))[2..6], #6 bits#
#if new s pointer > 16#
#on Inverse passed first 16 bits, active from [16,31] #

```

```

high_low_flag = new_s GE_U b'10000'.

```

- 569 -

```

#forward#
    fifo_not_full = CASE fifo_full
    OF ok_fifo: write
    ELSE read
    ESAC,

    fifo_write = CASE high_low #type change#
    OF high: write
    ELSE CASE flush_buffer #flush buffer when frame finished#
    OF twrite #needs 2 cycles to clear#
    ELSE CASE DFF(boot)(ck, reset, flush_buffer, i)
    OF twrite
    ELSE read
    ESAC
    ESAC,
    #from inverse#

    data_ready = CASE fifo_empty
    OF ok_fifo: write
    ELSE read
    ESAC,

    load_low = CASE reset #load low on reset to start things#
    OF rst: write,
    no_rst: PULSE(ck, reset, CASE (high_low_flag, data_ready) #load low word#
    OF (i, write): write
    ELSE read
    ESAC)

```

- 570 -

```

ELSE read
ESAC,
#delay reset for s and load_high#
reset_s = DFF(!_reset)(ck,reset,reset_rst).

load_high = CASE reset_s #load high next#
OF
rst:write,
no_rst:PULSE(ck,reset, CASE (high_low_flag,data_ready) #load high word#
OF
(!write):write
ELSE read
ESAC)
ELSE read
ESAC,

fifo_read = CASE load_low #read control for data_in FIFO#
OF
write:read
ELSE CASE load_high
OF
write:read
ELSE write
ESAC
ESAC,

#control signals#

(write_low,write_high) = CASE direction
OF
forward:{2}fifo_not_full
ELSE (load_low,load_high)
ESAC,

(high_out_sel,low_out_sel) = CASE direction
OF
forward:CASE high_low

```

- 571 -

OF high(left,right)
 ELSE (right,left)
 ESAC

ELSE {2}CAST(r_mund){s GE_U b"10000"}
 ESAC.

JOIN

```

(shift[17..32],fifo_in_dir_sel)    ->high_in,
(shift[1..16],fifo_in_dir_sel)     ->low_in,
(high_word,low_word,high_out_sel)   ->high_out,
(low_word,high_word,low_out_sel)    ->low_out,
(ck,reset,write_low,low_in,ZERO{16}b"0") ->low_word,
(ck,reset,write_high,high_in,ZERO{16}b"0") ->high_word,
(ck,reset,fifo_not_full,CASE high_low_flag
  OF t_high
  ELSE low
  ESAC,low)    ->high_low,

(ck,CASE forward
  OF forward,reset
  ELSE reset s
  ESAC,CASE direction
  OF forward,fifo_not_full

```

```

ELSE data_ready
  ESAC, new_s_ZERO(5)b'0') ->s.

```

```

OUTPUT (low_word,low_out,high_out,s,fifo_read,fifo_write)
END.

```

```

#the HUFFMAN decode/encode function#

```

```

#a pulse generator, glitch free#

```

```

FN PULSE = (bool:ck,!_reset:_reset,!_load:in) ->!_load:
CASE (in,DFF(!_load)(ck,_reset,in,read))
OF (write,read):write
ELSE read
ESAC.

```

```

FN SHIFT32_16 = (STRING[32]bit:buffer,STRING[5]bit:s) ->STRING[16]bit:
#left justified value, s shift const#
BEGIN
LET shift = ($ AND_B b'01111'7)[2..5]. #Input values rotated so always shift<16#
OUTPUT
CAST(STRING[16]bit)([INT j=1..16] MX16(CAST(STRING[16]bit)([INT i=1..16]buffer[(j-1+i)*shift])))
END.

```

```

FN SHIFT16X16_32 = (STRING[16]bit:n, STRING[4]bit:sel) ->STRING[32]bit:
BEGIN
LET sel_mux4= CASE sel[1..2]
OF b'00':sel[3..4]
ELSE b'11'

```

```

    ESAC,
    sel_mux4_high = CASE sel[1..2]
    OF   b'11"sel[3..4]
    ELSE b'00"
    ESAC,
    sel_mux8 = CASE sel[1]
    OF b'0: sel[2..4]
    ELSE b'111"
    ESAC,
    sel_mux8_high = CASE sel[1]
    OF b'1: sel[2..4]
    ELSE b'000"
    ESAC,
    OUTPUT CAST(STRING(32bit))
    MX_4(bit)(n[1],o[1],o[1],o[1],o[1],CAST([2]bool)sel_mux4),
    MX_4(bit)(n[2],n[1],o[2],o[2],o[2],CAST([2]bool)sel_mux4),
    MX_4(bit)(n[3],n[2],n[1],o[3],CAST([2]bool)sel_mux4),
    MUX_8(bit)(n[4],n[3],n[2],n[1],o[4],o[4],o[4],o[4],CAST([3]bool)sel_mux8),
    MUX_8(bit)(n[5],n[4],n[3],n[2],n[1],o[5],o[5],o[5],o[5],CAST([3]bool)sel_mux8),
    MUX_8(bit)(n[6],n[5],n[4],n[3],n[2],n[1],o[6],o[6],o[6],o[6],CAST([3]bool)sel_mux8),
    MUX_8(bit)(n[7],n[6],n[5],n[4],n[3],n[2],n[1],o[7],CAST([3]bool)sel_mux8),
    MX16(CAST(STRING(8bit))((INT i=1..8)n[9-i]) CONC ALL SAME(8B TO S o[8],sel[1..4]),
    MX16(CAST(STRING(9bit))((INT i=1..9)n[10-i]) CONC ALL SAME(7B TO S o[9],sel[1..4]),
    MX16(CAST(STRING(10bit))((INT i=1..10)n[11-i]) CONC ALL SAME(6B TO S o[10],sel[1..4]),
    MX16(CAST(STRING(11bit))((INT i=1..11)n[12-i]) CONC ALL SAME(5B TO S o[11],sel[1..4]),
    MX16(CAST(STRING(12bit))((INT i=1..12)n[13-i]) CONC ALL SAME(4B TO S o[12],sel[1..4]),
    MX16(CAST(STRING(13bit))((INT i=1..13)n[14-i]) CONC ALL SAME(3B TO S o[13],sel[1..4]),
    MX16(CAST(STRING(14bit))((INT i=1..14)n[15-i]) CONC ALL SAME(2B TO S o[14],sel[1..4]),

```

```

MX16(CAST{STRING[16]bit}((INT i=1..15)n[16-i])CONC o[15]),sel[1..4]),
MX16(CAST{STRING[16]bit}((INT i=1..16)n[17-i]),sel[1..4]),
MX16(CAST{STRING[16]bit}(b'0' CONC ((INT i=1..15)n[17-i]),sel[1..4]),
MX16(ZERO[2]b'0' CONC CAST{STRING[14]bit}((INT i=1..14)n[17-i]),sel[1..4]),
MX16(ZERO[3]b'0' CONC CAST{STRING[13]bit}((INT i=1..13)n[17-i]),sel[1..4]),
MX16(ZERO[4]b'0' CONC CAST{STRING[12]bit}((INT i=1..12)n[17-i]),sel[1..4]),
MX16(ZERO[5]b'0' CONC CAST{STRING[11]bit}((INT i=1..11)n[17-i]),sel[1..4]),
MX16(ZERO[6]b'0' CONC CAST{STRING[10]bit}((INT i=1..10)n[17-i]),sel[1..4]),
MX16(ZERO[7]b'0' CONC CAST{STRING[9]bit}((INT i=1..9)n[17-i]),sel[1..4]),
MX16(ZERO[8]b'0' CONC CAST{STRING[8]bit}((INT i=1..8)n[17-i]),sel[1..4]),
MUX_8[bit](b'0,n[16],n[15],n[14],n[13],n[12],n[11],n[10],CAST{[3]boolsel_mux8_high},
MUX_8[bit](b'0,b'0,n[16],n[15],n[14],n[13],n[12],n[11],CAST{[3]boolsel_mux8_high},
MUX_8[bit](b'0,b'0,b'0,n[16],n[15],n[14],n[13],n[12],CAST{[3]boolsel_mux8_high},
MUX_8[bit](b'0,b'0,b'0,b'0,n[16],n[15],n[14],n[13],CAST{[3]boolsel_mux8_high},
MX_4[bit](b'0,n[16],n[15],n[14],CAST{[2]boolsel_mux4_high},
MX_4[bit](b'0,b'0,n[16],n[15],CAST{[2]boolsel_mux4_high},
MX_4[bit](b'0,b'0,b'0,n[16],CAST{[2]boolsel_mux4_high},
b'0
)
END.
MAC REV_4 = (STRING[4]bit:in) ->STRING[4]bit:CAST{STRING[4]bit}((n[4],n[3],n[2],n[1])).

```

#in is data from bus, fifo_empty is input fifo control#
 FN HUFFMAN_DECODE = (mode:mode,STRING[2]bit:token_length_in,STRING[32]bit:buffer,STRING[5]bit:s)

- 575 -

```

->[bit_1_input,STRING[2]bit#token#]:

BEGIN
    MAKESHIFT32_16:input_decode.
COM
LET mag_out2 = CASE input_decode[9..12]
    OF b'1111':(input_decode[13..16] ADD_U b'10110') #add 22 to give value#
    ELSE input_decode[9..12] ADD_U b'00111' #add 7 to give value#
    ESAC,
MOC
LET sel_9_12 = CASE input_decode[9..12]
    OF b'1111':1
    ELSE 1
    ESAC,
mag_out2 = CASE sel_9_12
    OF 1:REV_4 input_decode[13..16]
    ELSE REV_4 input_decode[9..12]
    ESAC ADD_U
CASE sel_9_12
    OF: b'10110' #add 22 to give value#
    ELSE b'00111' #add 7 to give value#
    ESAC,
mag_out_huff=CASE input_decode[1]
    OF b'0:input/0
    ELSE CASE input_decode[3]
        OF b'1:input/1
        ELSE CASE input_decode[4]
            OF b'1:input/2
            ELSE CASE input_decode[5]
                OF b'1:input/3

```


- 576 -

```

ELSE CASE input_decode[6]
  OF b'1:input/4
    ELSE CASE input_decode[7]
      OF b'1:input/5
        ELSE CASE input_decode[8]
          OF b'1:input/6
            ELSE (S TO IN (b'0000" CONC mag_out2))[2]
              ESAC
            ESAC
          ESAC
        ESAC
      ESAC
    ESAC
  ESAC,

```

```

#on lpf still bit 1 is the sign bit#
sign = CASE mode
  OF lpf_still:input_decode[1]
    ELSE CASE mag_out_huff
      OF input/0:b'0
        ELSE input_decode[2]
          ESAC
        ESAC,

```

```

#select huff value, 0 (in lpf_send) or real value, rearrange the bits for real data#
#on lpf still bit 1 is sign bit so discard#
mag_out = CASE mode
  OF lpf_still:(S TO IN (CAST(STRING[9]bit)[INT j=1..9]input_decode[11-j]))[2]
    ELSE mag_out_huff
      ESAC,

```

- 577 -

```
token_length = b'000'CONC token_length_in,
```

```
#decode token, valid only during a token cycle#
```

```
token = CASE token_length[4..5]
```

```
OF b'10':input_decode[1..2],
```

```
   b'01':input_decode[1] CONC b'0'
```

```
ESAC.
```

```
JOIN (buffer,s) ->input_decode.
```

```
OUTPUT (sign,mag_out,token)
```

```
END.
```

```
#the huffman encoder#
```

```
FN HUFFMAN_ENCODE = (t_input: value, bit: sign, STRING[2]bit: token, l_mode: mode, l_cycle: cycle,  
  STRING[16]bit: buffer, STRING[5]bit: s)
```

```
-> (STRING[32]bit):
```

```
BEGIN
```

```
  MAKE SHIFT16X16_32: shiftL
```

```
  #encode value#
```

```
  LET header = CAST(STRING[2]bit)(b'1,sign),
```

```
  value_bit = CAST([16]bit)(IN_TO_S(16) value)[2],
```

```
  sub_const = CASE value
```

```
  _OF input/(7..21):b'00111',
```

```
      input/(22..37):b'10110'
```

```
      ELSE b'00000'
```

```
ESAC,
```

- 578 -

sub_value = ((IN_TO_S(input_exp)value)[2] SUB_U sub_const)[8..11],

enc_value =

CASE cycle

OF token_cycle: token CONC ZERO{14}b"0", #token is msb, max 2 bits#

data_cycle: CASE mode

#on intra & LPF pass thro value as 16 bit word, and reverse bit order, place sign first next to lsb#

OF lpf_still: CAST{STRING{1}bit} sign CONC CAST{STRING{15}bit} ((INT [-1..15]value_bit[17-1])

#otherwise value is to Huffman encoded, so out 16 bit as this is the max, the shift removes the extra bits#

ELSE CASE value

OF input/0: b"0" CONC ZERO{15}b"0",

input/1: header CONC b"1" CONC ZERO{13}b"0",

input/2: header CONC b"01" CONC ZERO{12}b"0",

input/3: header CONC b"001" CONC ZERO{11}b"0",

input/4: header CONC b"0001" CONC ZERO{10}b"0",

input/5: header CONC b"00001" CONC ZERO{9}b"0",

input/6: header CONC b"000001" CONC ZERO{8}b"0",

input/(7..21): header CONC b"000000" CONC (REV_4 sub_value) CONC ZERO{4}b"0", #sub 7 to give value#

input/(22..37): header CONC b"00000001111" CONC (REV_4 sub_value) #sub 22 to give value#

ELSE header CONC b"0000000111111111"

ESAC

ESAC,

skip_cycle: ZERO{16}b"0" #dummy value#

ESAC.

```
JOIN (buffer_enc_value,s[2..5]) ->shift.
```

```
#max value is 37 so 8 bits enough#
OUTPUT shift
END.
```

```
# some basic macros for the convolver, assume these will#
#be synthesised into leaf cells#
```

```
MAC MX_4(TYPE ly)=(ly:in1 in2 in3 in4, [2]boot:sel) ->ly:
```

```
CASE sel
OF (f,f):in1,
   (f,f):in2,
   (f,f):in3,
   (f,f):in4
ESAC.
```

```
MAC ENCODE4_2 = (l_mux4:in) ->[2]boot:
```

```
CASE in
OF uno:(f,f),
   dos:(f,f),
   tres:(f,f),
   quatro:(f,f)
ESAC.
```

```
MAC ENCODE3_2 = (l_mux3:in) ->[2]boot:
```

```
CASE in
OF t:(f,f),
   c:(f,f),
```

- 580 -

r(1,0)
ESAC.

MAC_MUX_3(TYPE t)=(t.in1.in2.in3, t_mux3.sel) -> t:
MX_4(t)(in1.in2.in3.in1, ENCODE3_2.sel).

MAC_MUX_4(TYPE t)=(t.in1.in2.in3.in4, t_mux4.sel) -> t:
MX_4(t)(in1.in2.in3.in4, ENCODE4_2.sel).

MAC_MUX_2(TYPE t)=(t.in1.in2, t_mux2.sel) -> t:
CASE sel
OF left:in1,
right:in2
ESAC.

MAC_MUX_8(TYPE ty)=(ty.in1.in2.in3.in4.in5.in6.in7.in8, {3}bool.sel) -> ty:
CASE sel
OF ((f,f).in1,
((f,f).in2,
((f,f).in3,
((f,f).in4,
((f,f).in5,
((f,f).in6,
((f,f).in7,
((f,f).in8
ESAC.

MAC_MX16=(STRING[16]bit.in, STRING[4]bit.sel) -> bit:
CASE sel
OF b"0000":in[1],
b"0001":in[2],

```

b'0010':in[3],
b'0011':in[4],
b'0100':in[5],
b'0101':in[6],
b'0110':in[7],
b'0111':in[8],
b'1000':in[9],
b'1001':in[10],
b'1010':in[11],
b'1011':in[12],
b'1100':in[13],
b'1101':in[14],
b'1110':in[15],
b'1111':in[16]
ESAC.
COM
MAC MX16=(STRING[16]bit:in, STRING[4]bit:sel) ->bit:
MUX 2[bit](
  MUX 8[bit](in[1],in[2],in[3],in[4],in[5],in[6],in[7],in[8],CAST([3]bool)sel[2..4]),
  MUX 8[bit](in[9],in[10],in[11],in[12],in[13],in[14],in[15],in[16],CAST([3]bool)sel[2..4]),
  CASE sel[1]
  OF b'0:left
  ELSE right
  ESAC).
MOC
MAC INT_BOOL = (!_quant:q) ->[3]bool:
CASE q
OF quant/0:(f,f,f),
   quant/1:(f,f,f),
   quant/2:(f,f,f),

```

```

quant/3:(1,1),
quant/4:(1,1),
quant/5:(1,1),
quant/6:(1,1),
quant/7:(1,1)
ESAC.

```

```

COM
MAC_MUX_3(TYPE t)=(t:in1 in2 in3, t_mux3:sel) ->t:
CASE sel
OF t:in1,
   c:in2,
   r:in3
ESAC.

```

```

MAC_MUX_4(TYPE t)=(t:in1 in2 in3 in4, t_mux4:sel) ->t:
CASE sel
OF uno:in1,
   dos:in2,
   tres:in3,
   quatre:in4
ESAC.
MOC

```

```

FN NOT = (bool:in) -> bool:CASE in OF t,f,h ESAC.

```

```

FN XOR = (bool: a b) -> bool:
CASE (a,b)
OF (t,f),(f,t):f
ELSE t
ESAC.

```

```

FN AND = (bool: a b) -> bool:
CASE (a,b)
OF (t,t).t,
   (f,bool)||(bool,f).f
ESAC.

```

```

FN OR = (bool: a b) -> bool:
CASE (a,b)
OF (t,f).t,
   (f,bool)||(bool,f).f
ESAC.

```

```

MAC DEL(TYPE t) = (t) -> t: DELAY(?t, t).

```

```

#a general d latch#
MAC LATCH (TYPE t) = (t_load: load, t_in) -> t:
BEGIN
  MAKE DEL(t): del.
  LET out = CASE load
    OF write: in
      ELSE del
    ESAC.
  JOIN out -> del.
  OUTPUT out
END.

```

```

#a general dff#
MAC DFF1 (TYPE t) = (bool: ck, t_in) -> t:
BEGIN
  MAKE DEL(t): del.

```



```

JOIN in->del.
OUTPUT del
END.

```

```

#a resetable DFF, init value is input parameter#
MAC DFF_INIT(TYPE i)=(bool:ck,t_reset:reset,t_load:load,t_in_init_value) ->t:
BEGIN
  MAKE DEL(i):del.
  LET out=CASE (load,reset)
    OF (write,t_reset):in,
      (read,rst):init_value
    ELSE del
    ESAC.
  JOIN out->del.
  OUTPUT CASE reset
    OF rst:init_value
    ELSE del
    ESAC
  END.

```

```

#a dff resetable non-loadable dff#
MAC DFF(TYPE i)=(bool:ck,t_reset:reset,t_in_init_value) ->t:
BEGIN
  MAKE DEL(i):del.
  JOIN in->del.
  OUTPUT CASE reset
    OF rst:init_value
    ELSE del
    ESAC
  END.

```

```

MAC PDEL(TYPE i,INT n) = (i:in) ->t
IF n=0 THEN DEL(i:in)
ELSE PDEL(i,n-1) DEL(i) in
FI.

```

```

MAC PDF1(TYPE i,INT n) = (bool:ck,t:in) ->t
IF n=0 THEN DF1(i)(ck,in)
ELSE PDF1(i,n-1)(ck,DF1(i)(ck,in))
FI.

```

#generates the new _mode from the old, and outputs control signals to the tokeniser#

```

FN MODE_CONTROL = (bool:ck, t_reset:reset, t_intra:intra_inter, bool:ipf_done,[7]bool:flags,
STRING[2]bit:token_in,t_octave:octave,t_state:state,t_direction:direction,t_load:load_mode_in
,t_cycle:cycle)
->(t_mode,t_mode,STRING[2]bit,t_diff,STRING[2]bit,t_mode):
#new_mode,proposed mode,current token,difference,token_length,#
BEGIN

```

```

MAKE [4]DFF_INIT(i_mode):mode,
DFF_INIT(i_diff):diff_out,
DFF_INIT(i_mode):next_mode.
LET nzflag=flags[1],
origin=flags[2],
noflag=flags[3],
ozflag=flags[4],
motion=flags[5],
pro_new_z = flags[6],
pro_no_z = flags[7].

```

- 586 -

lpf_done_del = DIFF{bool}{ck,reset,lpf_done,f}. #synchronise mode change at end of LPF#

LET next = (SEQ

#the proposed value for the mode at that octave, flags etc will change this value as necessary#
#proposed, or inherited mode from previous tree#

```

VAR  pro_mode:= CASE reset
    OF rst:CASE intra_inter #reset on frame start, so do lpf#
      OF intra:lpf_still
        ELSE lpf_send
          ESAC
      ELSE CASE lpf_done_del
        OFt:CASE intra_inter #store default mode in mode[4]#
          OF intra:still
            ELSE send
              ESAC

```

```

ELSE CASE state
  OF down1:mode[3], #jump sideways in ocl/1#
    / up0:mode[4]
  ELSE CASE octave
    OF ocl/0:mode[1],
      ocl/1:mode[2],
      ocl/2:mode[3]
      ESAC
    ESAC
  ESAC

```

```

ESAC,
new_mode:=pro_mode, #inherit the previous mode#
token_out:=b"00",

```

- 587 -

```

difference:=notiff,
token_length:=b'00',
flag:=f;
CASE direction
OF forward:

CASE pro_mode
OF void: CASE ozflag
OF tnew_mode:=stop
ELSE
ESAC,      #stay in these modes until end of tree#

void_still: ,      #intra so must zero out all of tree#

still_send:(token_length:=b'01';
CASE (nzflag OR pro_new_z)
OF t(token_out:=b'00';
CASE ozflag
OF tnew_mode:=stop
ELSE new_mode:=void
ESAC)
ELSE (token_out:=b'10';
new_mode:=still_send)
ESAC
),

send: CASE ozflag
OF t:(token_length:=b'01';

CASE (nzflag OR pro_new_z)

```

```

OF t:(token_out:=b'00";
  new_mode:=stop)
ELSE (token_out:=b'10";
  new_mode:=still_send)
ESAC
)
ELSE (token_length:=b'10";

CASE ( (NOTnoflag OR motion) AND NOTnzflag)
OFT( CASE origin
  OF tflag:=pro_new_z
  ELSE (flag:=pro_no_z;
    difference:=diff)
  ESAC;
CASE flag
OFT:(token_out:=b'10";
  new_mode:=void)
ELSE CASE origin
  OF t:(token_out:=b'01";
    new_mode:=still_send)
  ELSE (token_out:=b'11";
    new_mode:=send)
  ESAC
ESAC)
ELSE

CASE (motion OR origin)AND nzflag
OFT:(token_out:=b'10";
  new_mode:=void)
ELSE (token_out:=b'00";
  new_mode:=stop)

```

- 589 -

ESAC
ESAC
)
ESAC,

```
still: (token_length=b'01";
CASE (nzflag OR pro_new_z)
OF: (token_out=b'00";
      new_mode:=vold_still) #zero out tree#
ELSE (token_out=b'10";
      new_mode:=still)
ESAC
),
```

```
(tpf_still): (token_out=b'00";
              token_length=b'00");
```

#for ELLA only DUMB!!!#

```
(tpf_send): (difference:=diff;
             token_length=b'01";
```

```
CASE (noflag OR pro_no_z)
OF t: (token_out=b'00";
      new_mode:=tpf_stop)
ELSE (token_out=b'10";
      new_mode:=tpf_send) #as mode stop but for this block only#
ESAC)
```

ESAC,

inverse:

CASE pro_mode

```

OF void: CASE ozflag
  OF t:new_mode:=stop
  ELSE
  ESAC,

void_still:,

send: CASE ozflag
  OF t:(token_length:=b'01'; //repeat of still-send code#
    CASE token_in[1]
    OF b'1:new_mode:=still_send,
      b'0:new_mode:=stop
    ESAC
    )
  ELSE (token_length:=b'10';
    CASE token_in
    OF b'11': (difference:=diff;
      new_mode:=send),
      b'01':new_mode:=still_send,
      b'10':new_mode:=void,
      b'00':new_mode:=stop
    ESAC
    )
  ESAC,

still_send: (token_length:=b'01';
  CASE token_in[1]
  OF b'1:new_mode:=still_send,
    b'0: CASE ozflag
      OF t:new_mode:=stop

```

- 591 -

```

ELSE new_mode:=void
ESAC
ESAC
).
still: (token_length:=b'01";
CASE token_in[1]
OF b'1:new_mode:=still,
   b'0:new_mode:=void_still
ESAC
),
(lp_send):(difference:=diff,
token_length:=b'01";
CASE token_in[1]
OF b'0:new_mode:=lpf_stop,
   b'1:new_mode:=lpf_send
ESAC).
lpf_still:
ESAC
ESAC;
OUTPUT (new_mode,pro_mode,token_out,difference,token_length)
).
LET load_mode = CASE (reset,lpf_done_del) #store base mode in mode[3]& mode[4], base changes after lpf#
OF (rst,boof)|(ft reset,1):(read,read,write,write)
ELSE CASE (octave,load_mode_in)

```



```

OF (oct/1,write):(write,write,read,read),
  (oct/2,write):(read,write,write,read)
ELSE (read,read,read,read)
ESAC
ESAC.
#save the new mode& difference during a token cycle, when the flags and tokens are valid#
JOIN (ck,reset,CASE cycle
OF token_cycle:write
ELSE read
ESAC, next[1],still) -->next_mode,
-->diff_out.

(ck,reset,CASE cycle
OF token_cycle:write
ELSE read
ESAC, next[4],nodiff)

#now write the new mode value into the mode stack at end of cycle, for later use #
FOR INT 1=1..4 JOIN (ck,no_rst,load_mode[i],CASE (reset,lpf_done_del)
OF (no_rst,1):(rst,bool):next[2]
ELSE next_mode
ESAC,still) -->mode[i].
#dont update modes at tree base from lpf data, on reset next[1] is undefined#

OUTPUT (next_mode,next[2],next[3],diff_out,next[5],next[1])
END.

#the tree coder chip#
#threshold = 2*quant_norm#

FN PALMAS= (bool:ck,1_reset:reset,1_direction:direction,1_intra:intra_inter,1_channel_factor:channel_factor,

```

```

[4]t_quant:=quant_norm, STRING(16)bit:buffer_in,
t_input:=new_old,[4]t_result:=threshold,t_fifo:=fifo_full_fifo_empty, STRING(xsize)bit:col_length,
STRING(yysize)bit:row_length, STRING(xsize)bit:ximage_string, #ximage#
STRING(yysize)bit:yimage_string, STRING(11)bit:yimage_string_3#yimage&yimage*2.5#)
->(!_input,!_spare_addr,!_load,!_cs),(t_load,!_cs),STRING(16)bit[2]t_load,bool,!_cycle):

```

```

#old,address,(rw_new,cs_new),(rw_old,cs_old),buffer_out,fifo_read_fifo_write,cycle#

```

```

BEGIN

```

```

    MAKEDECIDE:=decide,

```

```

    ADDR_GEN:=addr_gen,

```

```

    HUFFMAN_ENCODE:=huffman_encode,

```

```

    FIFO_BUFFER:=fifo_buffer,

```

```

    HUFFMAN_DECODE:=huffman_decode,

```

```

    MODE_CONTROL:=mode,

```

```

    CONTROL_COUNTER:=control_counter,

```

```

    BLK_SUB_COUNT:=sub_count,

```

```

    OFF_INIT(!_channel):channel,

```

```

    QUANT:=quant.

```

```

LET

```

```

    nzflag:=decide[1],

```

```

    origln:=decide[2],

```

```

    noflag:=decide[3],

```

```

    ozflag:=decide[4],

```

```

    motion:=decide[5],

```

```

    pro_no_z:=decide[7],#pro_no_z or pro_new_z#

```

```

    pro_new_z:=decide[6],

```

```

new_mode = mode[1],
pro_mode = mode[2],
token_out = mode[3],
difference = mode[4],
token_length = mode[5],

pro = quant[1], #pro no, or pro_new#
lev_out = (S_TO IN quant[2])[2], #corresponding level#
sign = quant[3], #and sign #

octs = addr_gen[2],
sub_en = addr_gen[3],
lree_done = addr_gen[4],
lpf_done = addr_gen[5],
state = addr_gen[6],

cycle = control_counter[2],
cs_new = control_counter[7],
rw_new = read,
rw_old = control_counter[8],
cs_old = control_counter[9],

load_channel = CASE (sub_en, sub_count[2]) #change channel#
  OF (1,1): write
  ELSE read
  ESAC,

new_channel = CASE channel_factor
  OF luminance.y
  ELSE CASE channel
    OF y:u,

```

- 595 -

```

    u.v,
    v.y
    ESAC
    ESAC,
    #flush the buffer in the huffman encoder#
    flush_buffer = DFF(bool){ck,reset,CASE channel_factor
    OF luminance:CASE load_channel
    OF write:t
    ELSE f
    ESAC,
    color: CASE (channel,load_channel)
    OF (v,write):t
    ELSE f
    ESAC
    ESAC,f),

```

```

frame_done = PDF1{bool,1}{ck,flush_buffer},

```

```

fifo_write=fifo_buffer[6],
fifo_read=fifo_buffer[5],
s=fifo_buffer[4],

```

```

buffer_out = fifo_buffer[1],

```

```

lev_in = huffman_decode[2],
sign_in = huffman_decode[1],
token_in = huffman_decode[3],

```

```

del_new = PDF1{input,4}{ck,new},

```

```

#old has variable delays for inverse#
del_old = CASE (direction,pro_mode)
  OF (forward,l_mode){(inverse,send|still_send|pf_send|void): PDF1(t_input,4)(ck,old)
  ELSE PDF1(t_input,1)(ck,old)
  ESAC,
decide_reset=CASE reset
  OF rstst
  ELSE control_counter[3]
  ESAC,

oct_sel = CASE pro_mode
  OF pf_still|pf_send|pf_stop:qualro
  ELSE CASE (octs,channel)
    OF (oct/0,y):uno,
      (oct/1,y)|(oct/0,u|v):dos,
      (oct/2,y)|(oct/1,u|v):tres
    ESAC
  ESAC,

threshold_oct = MUX_4(t_result)(threshold[1],threshold[2],threshold[3],threshold[4],oct_sel),

quant_oct = MUX_4(t_quant)(quant_norm[1],quant_norm[2],quant_norm[3],quant_norm[4],oct_sel).

JOIN (ck,decide_reset,threshold_oct,new,old,threshold_oct,threshold_oct,control_counter[6])->decide,
(ck,reset,intra_iter,pf_done,decide,token_in,octs,state,direction,control_counter[1],cycle)->mode,

#delay the new&old values by 5 or 1 depending on mode & direction#
((IN TO S(input_exp|del_new)[2], (IN TO S(input_exp|del_old)[2],
  (IN TO S(input_exp|lev_in)[2], sign_in,direction,quant_oct,difference,pro_mode) ->quant,

```

- 597 -

```

(ck,reset,new_channel,channel,load_channel,sub_count[1],col_length,row_length,
ximage_string,yimage_string,yimage_string_3,control_counter[4],control_counter[5],new_mode)->addr_gen,

(ck,reset,direction,cycle,pro_mode,lev_out,huffman_decode[2],buffer_in,fifo_full,
fifo_empty,huffman_encode,tokens_length,flush_buffer,quant_norm[4])    ->fifo_buffer,

(lev_out,sign,tokens_out,pro_mode,cycle,fifo_buffer[2],s)    ->huffman_encode,

(pro_mode,tokens_length,fifo_buffer[2] CONC fifo_buffer[3],fifo_buffer[4])    ->huffman_decode,

(ck,reset,sub_en,i,i)    ->sub_count,

(ck,reset,pro_mode,new_mode,direction)    ->control_counter,

(ck,reset,load_channel,new_channel,y)    ->channel.

OUTPUT

(CASE new_mode
OF void|void_still:input/0
ELSE (S_TO_INpro)[2]
ESAC    ,addr_gen[1],(rw_new,cs_new),(rw_old,cs_old),buffer_out,(fifo_read,fifo_write),frame_done,cycle)
END.

COM
#the decoder for the barrel shifter- decides if the bit value and q value are #
#in the upper-triangle, or diagonal and set the control bits    #

MAC DECODE(INT n) = (l_quant,q)    ->[qmax](bool#upper diag#,bool#diagonal#):
BEGIN
    #one bit of the decoder#
    MAC DECODE_BIT(INT j)= (l_quant,q)    ->(bool,bool):
    CASE q

```

```

OF quant/(0..qmax-j):(1,0), #upper-triangle#
  quant/(qmax-j+1):(1,0) #diagonal#
ELSE (1,0)
  ESAC.
OUTPUT((INT j=1..qmax)DECODE_BIT(j)(q))
END.

#now the selector fn to mux between the data_in bit 0 or 1 depending on q#
MAC_SELECTOR = (1..quant;q,STRING(INT n)bit:data)
->(STRING(n)bit#level#,STRING(n)bit#round_level#):

BEGIN
  #the 3->2 bit selector#
  MAC_SELECTOR_BIT = ((2)boot:upper_or_diag,bit:data) -->(bit,bit):#level[],round_level[]#
  CASE upper_or_diag
  OF (1,0):(data,data), #upper-triangle#
    (1,1):(b'0,b'0) #diagonal#
    ELSE (b'0,b'1) #lower-triangle#
  ESAC.
  MAKE DECODE(n):decode,
    [qmax]SELECT_BIT: select.
  JOIN (q) -->decode.

  FOR INT j=1..qmax JOIN (decode[],data[n-qmax+j]) -->select[j].

  OUTPUT (data[1..n-qmax] CONC (BIT_STRING(qmax){(INT j=1..(qmax))select[j](1)}), #level#
    data[1..n-qmax] CONC (BIT_STRING(qmax){(INT j=1..(qmax))select[j](2)}), #round_level#
  )
  END.
  MOC

  #now the selector fn to shift the level depending on q#

```

```

MAC BARREL_SHIFT_RIGHT = (l_quant:q.STRING(INT n)bit:data) ->(STRING(n)bit#level#):
MUX_8[STRING(n)bit]{
    data,
    b"0"CONC data[1..n-1],
    b"00"CONC data[1..n-2],
    b"000"CONC data[1..n-3],
    b"0000"CONC data[1..n-4],
    b"00000"CONC data[1..n-5],
    b"000000"CONC data[1..n-6],
    b"0000000"CONC data[1..n-7],
    INT_BOOL q.
}

```

#the bshift for the inverse, to generate the rounded level #

```
MAC BARREL_SHIFT_LEFT = (l_quant:q.STRING(INT n)bit:data#lev#) ->(STRING(n)bit#round_level#):
```

```

MUX_8[STRING(n)bit]{
    data,
    data[2..n]CONC b"0",
    data[3..n]CONC b"01",
    data[4..n]CONC b"011",
    data[5..n]CONC b"0111",
    data[6..n]CONC b"01111",
    data[7..n]CONC b"011111",
    data[8..n]CONC b"0111111",
    INT_BOOL q.
}

```

#the function to return the quantised level(UNSIGNED), and proposed value given, #

the new&old values, forw/inverse direction

```

FN QUANT = (STRING(input_exp)bit: new old lev_inv,bit:sign lev_inv, t_direct:direction,t_quant,q,t_diff:difference,
            t_mode:mode)

```


- 600 -

-> (STRING[input_exp]bit,STRING[input_exp]bit) #pro,lev& sign# :

BEGIN

LET

#decide which of new-old or new will be quantised, and the sign of the level#
#level is stored in sign & magnitude form#

dir_sel = CASE direction
OF forward:left,
inverse:right
ESAC,

sub_sel = CASE difference
OF diff:left
ELSE right #put old=0#
ESAC,

sub_in= MUX_2(STRING[input_exp]bit)(old,ZERO[input_exp]b'0',sub_sel).

no =ADD SUB_ST(new,sub_in,subtl).

lev_final= ABS_S no, #now input_exp+1 bits#

sgn_level = MUX_2(bit)(#sign of value to be quantised#
no[1],
sign_lev_inv,
dir_sel).